USB based attacks & security solutions

Mast, Fabian
Department Informatics
Technische Universität München
München, Germany
fabian.mast@tum.de

***Abstract*—The Universal Serial Bus (USB) has become widely adopted interface and is represented in nearly every modern computer system. However, the frequent use has exposed many devices to a wide spectrum of security vulnerabilities which lie within this protocol. To create awareness on these issues, this research paper presents a comprehensive and intuitive approach to the classification of USB-based attacks and elaborates the underlying security flaws within the USB specification, such as *Plug and Play* or missing firmware authentication. To understand exactly where and how mistakes have been made during the design process, we use exemplary attacks which represent each class, respectively. This allows individuals and cooperations to better assess the risks associated with USB usage and implement adequate security measurements. Furthermore, this paper proposes security counter measurements and evaluates them, offering to make informed decisions about them and to see, how to defend against such attacks. These include hardware-level protections, encryption mechanisms and software solutions. We address the need of change in newer USB specifications and identify design flaws and oversights that contribute to emerging these vulnerabilities.**

***Index Terms*—USB based attacks, classification, solutions**

I. Introduction

To interact with a PC, most of the time an external keyboard and mouse is used. Although in the early days of HID's (Human Interface Device), various kinds of interfaces for the connection were used, nowadays the most prominent protocol is USB. Options and improvements in usability such as automatic configuration of a driver, the ability to plug and unplug a device while the system is running (hot pluggable) and the integrated power supply have given the user an easy-to-use interface. Manufacturers benefit from cost-efficient production of devices because there is no complex intelligence for communication necessary in a USB device; the host takes over communication and computation. Finally, Universal Serial Bus (USB) offers a singular interface capable to connect almost any device, therefore it finds application from mobile storage to peripheral devices to USB-powered fans.

As the history of USB starts in 1996 with the introduction of USB 1.0, the protocol was introduced in Microsoft Windows 95. Following the integration to Windows 98, the number of vendors began to increase and at the same time, USB 1.1 improved compatibility and added support for hubs. To meet the increasing demand for faster buses, USB 2.0 was released in 2000, offering a data transfer rate of 480 Mbps at high speed. This met the requirements for most USB devices while maintaining backwards compatibility and thus became the most widely adopted version of USB in HID devices. In 2008, the USB Implementors Forum (USB-IF) released USB 3.0 (later called USB 3.1 Gen. 1) introducing SuperSpeed at a rate of 5Gbps, providing data transfers ten times faster than USB 2.0. Later, USB 3.1 Gen. 2 further enhanced the transfer rate to 10 Gbps, referred to as SuperSpeedPlus. With USB 3.2 and USB 4.0, which respectively double the transfer rate of their predecessor, there will be more advanced versions in the future. Note that up to the point of introducing USB Type-C, security measurements were not part of any USB specification. It is the first protocol that offers a security feature, namely the authentication protocol [13]. While there were improvements on speed of the buses and other factors such as supported power output and added USB types, the basic design decisions were not changed over the years. Fundamental components of the USB protocol like *trust-by-default*, non-encrypted communication and more represent a big attack surface to this interface. Adding to that, according to a study done by Tischer et al. [29] 45%–98% of "lost" USB flash drives get connected to a PC, the first device in this study was plugged in within 6 minutes. This highlights how dangerous it is to expose unsuspecting users to attacks in this realm. We show how these core principles of the USB protocol led to an overall insecure system as soon as unsuspecting insiders as well as outsiders can physically access a device with an USB interface. We further categorize arising attack vectors coming from these basic concepts up to the abstraction level of the Operating System (OS) and show the systematic behind each class based on one representative example. Finally, researched solutions to these inherent flaws of USB are presented, explained, and evaluated based on the proposed classes which allow to insights on how to implement security measures.

This paper presents the following contributions:

- Discussing the non-existence of adequate security measurements throughout all USB versions
- Categorizing well-known attacks by linking them to the inherent design decisions made during the conceptualization of the USB protocol
- Highlighting trade-offs between usability and security in that regard
- Explaining technical details used for attacks in this realm
- Proposing researched security concepts and evaluating them by the categorization made earlier

II. Background

For an application to interact with a USB device, the OS offers a fixed number of commands which originate from the interface of the chosen driver. The application itself is not in contact with technical details from the USB protocol. The communication from the viewpoint of the OS is therefore of main interest.

A. Differences in communication in USB versions

From USB 1.0, USB 2.0 enhanced the bus protocol with reduced transaction overhead while still using a shared bus architecture like USB 1.0. Therefore, it is backward compatible. As of USB 3.0, a new bus architecture is introduced with separate transmit and receive channels, which are represented by two separate data lines, and allows hubs to direct traffic the specified device. Enhanced SuperSpeed refers to devices supporting SuperSpeed and SuperSpeedPlus, therefore version USB 3.1 Gen. 1 and higher.

USB version differences		
Version	Speed name	max. data rate
USB 1.0	Low Speed	1,5 Mbit/s
USB 1.0	Full Speed	12 Mbit/s
USB 2.0	Hi-Speed	480 Mbit/s
USB 3.0/3.2 Gen 1	SuperSpeed	5 Gbit/s
USB 3.1/USB 3.2 Gen 2	SuperSpeedPlus	10 Gbit/s
USB 3.2 Gen 2x2	SuperSpeed20	20 Gbit/s
USB 4.0	USB 40	40 Gbit/s

Fig. 1. Different USB versions and protocol speeds

B. Root hub, Host controller

The root hub operates as source of all USB ports and collaborates with host controllers. As a software component, it is the first logical point of contact for any USB device and is involved from enumeration to communication. It is responsible for monitoring device status and configuration as well as power distribution over the USB system.
The host controller is a hardware component which formats the data and puts it on the bus and vice versa, mostly integrated into the motherboard. This is where a device is plugged in. It must detect devices, manage data flow, perform error checking, provide, and manage power and exchange data with devices. It is also the task of the host controller to ensure that every device can send and receive data and does not starve. We will refer to the combination of both as *host* throughout this paper.

C. Transfers

To interact with a device, the USB host always must initiate the transfer (except for remote wakeup signaling). The USB device then responds to the data sent. Data which is directed to a USB device is firstly saved in a buffer. This buffer has a logical address, which is called an endpoint. Every endpoint has an address from 0 to 15 for each of the following signals. It can receive an IN signal for transmitting data to the device (e.g. write on a flash drive) or an OUT signal for requesting data (e.g. reading data from a flash drive). It respectively responds with either a status signal or the requested data.

D. Transfer Types

There are four different ways to communicate on the protocol:

- Bulk
- Control
- Interrupt
- Isochronous

Bulk is commonly used in printers and flash drives where there is no need for fast reaction time of the device. It offers the fastest data transfer rates when the USB bus is idle. However, when the bus is busy, the host device will delay the transfer until the bus becomes idle again.
Control is a transfer type which is mandatory to be supported for every device. In enumeration, this is the standard protocol. If a guaranteed maximum latency is necessary, Interrupt is used. Mostly applied in mice and keyboards, the driver holds the information in the buffer and sends it on the bus as soon as an IN signal is received. Typically, this triggers an interrupt at the host system.
Isochronous transfer also ensures a maximum guaranteed latency but differs by limiting the response to a defined number of bytes in a specific interval. Primarily utilized in streaming audio or video, it is not necessary to support error correction.

E. Enumeration

On attaching, the host system recognizes a new USB device through voltage changes on the signal lines D+ and D- within the power system of the host. At first, the host sends a reset signal to remove a possible previous state and assigns an endpoint address of 0 for the time of enumeration. Then, by using multiple standard requests (Control Transfer) the hosts requests a series of data structures using a `GetDescriptor` request. Over several steps, the presence and values of attributes of the device are requested, including

- Device descriptor
- Configuration descriptor(s)
- Interface descriptor(s)
- Endpoint descriptor(s)
- Product ID string descriptor
- Serial Number string descriptor
- Vendor ID string
- Class descriptor (for speed agreements higher than High-Speed)

Devices can have more than one *function*, e.g. one USB connection can support a printer and a scanner. These devices are called *composite devices*. Different configuration descriptors and different interface descriptors are exchanged for each function. For each configuration descriptor there is at least one interface descriptor which binds an endpoint (address) with

a transfer type to an interface. One function is represented by a group of interfaces to support different transfer types for a function and is assigned to a device class, e.g. HID or Ethernet adapter. Note that speeds from USB 2.0 or lower are negotiated before the `GetDescriptor` request by using the power system and checking for specific voltage change.
With a `SetDescriptor` request, a mutual transfer protocol is established. With Windows, the Plug and Play (PnP) Manager locates the INF file which is needed to identify a driver according to vendor ID and product ID. The device is now ready to be used by applications over the API of the OS.

F. Bus speed decision and hubs

A USB device employs the highest mutually supported USB speed when connected to a hub. The hub communicates with its parent device at the highest common USB speed, even if the connected devices operate at a lower speed. A complete example is given in Fig. 2. In USB, the root hub is at level 0. Each hub, which is directly connected to the root hub is on level 1. The level increases incrementally with more hubs in between the newly connected hub and the root hub.

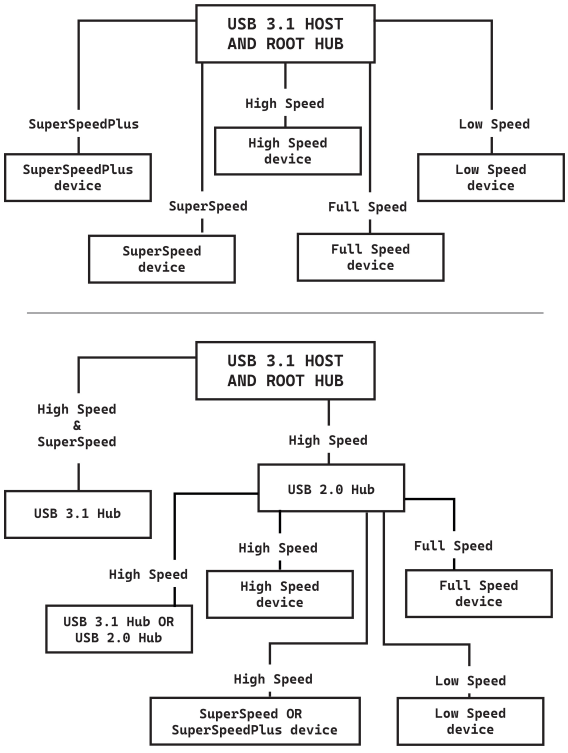

Fig. 2. Progression of USB speeds across different hubs and devices. Adapted from [5]

G. Physical construction

For instruction purposes the components and processes here listed are only referring to the USB 2.0 specifications. USB Type-A male pinouts consist of a set of four pins that define the electrical connections. In Fig. 3, Pin 1 is denoted as VBUS or +5V and serves as the source for +5V DC power supply to connected devices. Pin 2 corresponds to the Data- signal (D-), responsible for transmitting data from the device to the host. Pin 3 represents the Data+ signal (D+). The differencial pair only send the same signal during a reset process. Pin four functions as the ground (GND) reference.

Fig. 3. USB 2.0 Type-A male

To interact on the physical layer of the USB protocol, a USB controller is used. The main parts being used are a transceiver, a serial interface engine (SIE), buffers and registers. Directly connected to the USB connector is the transceiver. It only forwards information from the bus to the SIE. As the first layer of logic in a USB controller it decides which packets contain an endpoint address which corresponds with defined interfaces. Defined interfaces are interfaces enlisted during enumeration. Also, CRC checksums are checked and generated. To achieve clock synchronization at both ends of the channel, the protocol uses bit stuffing. This adds a 0 bit after six consecutive 1 bits. Encoding and decoding of data using NRZI (Not-Return-Zero-Inverted) with bit stuffing provides data to the transceiver and to buffers. Buffers will hold data that is received or ready to transmit. CPU and program memory are responsible for creating a communication channel on enumeration and other tasks the USB device is responsible for. The term *firmware* appears here as the code saved in the USB controllers memory. E.g., the device's functions can here be specified.

H. Drivers

A driver for a USB device gets decided for each function. By using the provided device class, the OS can then select an according driver and therefore offers any application to use this API. In recent years, the ability to block access to devices which are enlisted as microphone or camera has been added.

III. Related Work

Throughout this paper we will use examples of 29 typical USB based attacks introduced by Nissim [20]. In 2018, Tian et al. presented a classification and enlisting of attacks from which we will find inherent design flaws in the USB protocol [28]. At the end, we provide an overview over existing counter measurements covering our classification proposed by numerous papers. In contrast to the SoK "Plug & Pray" provided in 2018 we create a new classification approach and explain in detail the functionalities of a typical attack in this class to understand exactly where weak points in the specification are.

IV. Exploiting fundamental design choices in the USB protocol

We start by grouping USB based attacks into four different categories:

1) Modified Hardware: To attack in this area, specific hardware has to be implemented and used. Regular USB devices do not offer enough capabilities to execute them.

2) Host/OS attacks: Attacks are directed from a USB device to the host, directed to specific components. Drivers and vulnerabilities in how the OS handles a USB device can be targeted.
3) Unexpected software functionality: Through changes in the interface descriptors of a USB device it is possible for an USB flash drive to pose e.g. as a keyboard without the user noticing it. Through USB's versality the attack area is extremely broad.
4) USB device attacks: The plugged in USB device is here victim of the host system. Abusing devices and access them without restrictions is possible.

After framing USB's flaws, we categorize and discuss different attacks within a group. Afterwards, security counter measurements are proposed for each individual group in Section V.

A. Modified Hardware

Over the years a wide range of hardware-based attacks on the hardware side of the USB protocol has been discovered. Key issues are the missing encryption of traffic between USB devices and the presumption that USB devices always conform to the USB specifications. The upside to the usability of these properties are cheaper implementations of devices as well as less overhead in the communication. We will show later that the effect of the latter can be neglected.

1) Missing communication channel encryption: Besides of error checking codes and bit stuffing, pure application data is transferred on. For an eavesdropper, it is not difficult to read transmitted traffic as soon as he has gotten access to it. This opens the possibility of intercepting data using MitM or Man-on-the-Side attacks, but also allows other side channels of USB like electromagnetic fields or transfer types being used to read. [31] [22].

2) Omitting the consideration of non-standard behavior: With countless vendors for USB devices, one must hope that a newly plugged in piece of hardware will not malfunction. Lack of securing against incorrect implementations of the USB standard enables attacks targeted to destroy the system to which a device is plugged in. This DOS (Denial of Service) attack affects every device which has an USB interface, regardless of if powered up or not [30] [4].

3) Proof of concept: To illustrate an attack vector, we refer to Neugschwandtner et al. [19]. As specified in the USB 2.0 specifications (Fig. 4) traffic going downwards from a hub to a device is broadcasted to all devices and hubs connected to the hub using USB 2.0. Upstream traffic is not broadcasted and will only be received by the root hub and all hubs in between. Note that in hubs with versions higher than USB 2.0 the communication is directed to the specified device and this attack is not feasible.

Through the process of how the bus speed and thus the version of the USB protocol gets chosen, it is possible to eavesdrop downstream traffic (OUT packets) as soon as a USB 2.0 hub gets used. Regardless of if hubs or devices in ranks further away from the host use versions higher than USB 2.0,

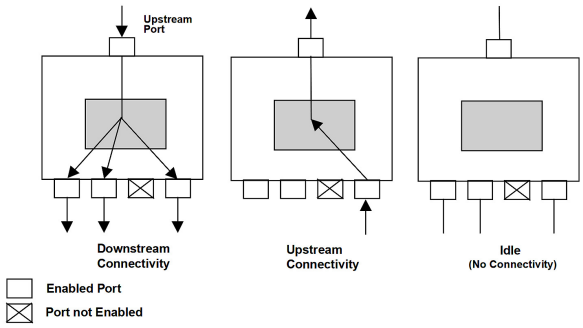

Fig. 4. Hub connectivity [25] Section 11.1.2.1

it is possible to read all traffic going through that hub. To add to that, input devices like keyboards, mice and fingerprint readers often are manufactured to communicate on USB 1.x slow speed due to the cheaper and sufficient data rate [25] (USB snooping made easy: Crosstalk). A SIE only accepts traffic directed to registered endpoint addresses and discards other traffic. In theory, it is possible to capture all data arriving at the data lines of the device by changing the behavior of the SIE. However, due to manufacturers implementing this part of the USB controller on regular devices in hardware, executing this kind of attack is not trivial and implementing a custom SIE can be expensive. [19]. Similar eavesdropping attacks have been done by measuring differences in voltage in an USB system [22]. A much easier method to implement for sniffing traffic is using a traffic logger which is plugged in between the hub one level lower than the target device. While logging e.g. keystrokes with a device such as [14] is technically like a Man-in-the-Middle attack and thus not very hard to implement, it is notable that this kind of device is less practical than the attack vector used before. Visual recognition of a device hooked in between is easier and while it can only sniff one device it has to be one preferably not often unplugged. We conclude that the foundation of every physical wiretap attacks the issue lies in the lack of proper communication channel encryption. To show the scope, e.g. IronKey promotes a USB flash drive securing sensitive data through encrypting it on the USB device [15]. This mechanism is undermined in the extent of this attack because encryption happens after transfer.

Another attack in this realm present USB Killers. A capacitor is charged by the power supply of the host and by discharging a high voltage pulse through the USB interface into the connected USB hub, it damages the underlying electronical system. The oversight here is the absence of protection measurements, such as overvoltage protection circuits. While a device can be seriously damaged this way, [4] shows that data stored on hard disks (SSD or HDD) remains intact after such an attack. Therefore, it falls in the category of DOS (Denial of Service) attacks. Even not powered up devices are affected, since hardware with integrated batteries is available [30].

B. Unexpected software functionality

Due to different USB design decisions, it is possible to emulate functions which an unsuspecting user would not expect and enable a big class of attack vectors. Many well-known attacks in this section are known under the category of BadUSB attacks, such as Rubber Ducky, Evilduino, USBdriveby and more [20] [21]. Note that though mostly USB drives or specific hardware components are used within BadUSB, an attacker can also deploy such hidden functionalities in e.g. a mouse [16] Fig. 4. While the autorun provided from Windows caused vulnerabilities by itself, running malicious code automatically as soon as connected in the past was fixed in Windows 7 [26].

1) Missing firmware authentication: The inability to prove whether a device is from a trusted source, (i.e. an official vendor), or not opens doors for anybody to create their own USB device which works without restrictions. While most USB devices on the market have been certified by the USB-IF, this step is not mandatory since only the right to use the USB logo and a specific Vendor-ID evolve through that process [3] page 32.

2) Plug and Play Property: Though Plug and Play (PnP) support is one of USB's most user-friendly feature, when not properly managed this is a key factor of Host/OS attacks. By not asking a user for validation if a device driver should automatically be assigned when a device is added for the first time, it also allows tampering while a computer is locked by a screen saver [7].

3) Composite devices: The concept of composite devices allows devices to provide a set of functionalities within one device. Commonly used in headsets or webcams, one USB connection can be sufficient for headphones, microphones, and cameras. Although a clear value to the customer, devices can act with unexpected functions such as a USB flash drive posing as a keyboard without the user noticing it [20] (Page 5).

4) Proof of concept: To illustrate the security significance of these design decisions, we explain a proof-of-concept attack where a USB thumb drive acts as a Gigabit-Ethernet controller. With two different device descriptors, the attached device enlists at the host in the enumeration process as 1) a USB thumb drive and 2) as an emulated DHCP (Dynamic Host Configuration Protocol) server. By setting the InterfaceClass field of one of the devices two device descriptors interface descriptor from 0x08 (for mass storage) to 0xEF (for miscellaneous), the InterfaceSubclass Field to 0x04 and the InterfaceProtocol field to 0x01, the device identifies itself via the "RNDIS (Remote Network Driver Interface Specification) over Ethernet" as a new Ethernet Adapter and simultaneously as a normal USB flash drive. By only adding a DNS (Domain Name System) server, where the malicious entry is added, and not setting a Default Gateway, the internet connection up to that point remains as is and only the DNS server is changed from the original one to the one on the USB device. To guarantee that the newly plugged in device is used, it shows as a Gigabit ethernet controller. Nohl states that "OSes prefer a wired network controller over a wireless one and a Gigabit ethernet controller over a slower one. This means the OS will use the new spoofed Gigabit controller as the default network card." [7]. The absent of a) verified firmware, b) PnP support and c) the ability for complement devices being used without the user knowing of what kind enable this MitM attack. By controlling the newly specified server, intercepting all network traffic is possible. To demonstrate the inconspicuousness Nohl also implements this attack on a rooted Android phone, pretending to be charged via USB cable [7].

C. Host/OS attacks

While software attacks showed up to this point have used legitimate USB device behaviors, we now investigate vulnerabilities arising through the trust the host puts on the device. The goal is to run arbitrary code or to access data stored on the host machine. We achieve this by forcing non-standard behaviors of USB devices to bypass restrictions and protection measurements which are made by the host or by abusing the absence of restrictions in the first place.

1) Inherent trust of the host system on plugged in devices: The missing focus of software developers allows a large spectrum of USB attacks. To better understand the dynamics and effects, three examples are elaborated. Like the PnP property it increases usability by making a device accessible in a fast manner.

a) Driver fuzzing: Fuzzing is the process of sending a large volume of random input, valid or invalid for the receiver, to find differences in how the program reacts based on this input [33]. With this and other techniques many flaws within USB drivers have been found over the years [2], e.g. using FaceDancer. While the attack vector in this example is connected to the missing firmware verification, it could not have been prevented from happening as we will see below. The implicit trust placed on the connected device is the root cause for an attack vector like this. Although we can fuzz the host, note that the USB property of putting all necessary intelligence onto the host is an advantage in USB. It is much easier to validate code running on the host OS than for manufacturers to search their firmware code for security vulnerabilities. By explaining the functionality of this attack below, we better understand where software related security issues, away from the actual USB protocol, start to appear.

b) Juice Jacking: Relating to the work of Lau et al. [17] in 2013, a wide spread vulnerability in IOS devices brought attention to the realm of Juice Jacking. As we see more publicly available USB ports, it is critical to understand what kind of trust you place on the wall charger in a mall by plugging your mobile phone in to charge it. Unsuspicious users may think that only the powering functionality is being used, therefore the D+ and D- pin are untouched. This is not the case. With the inherent trust placed on USB, it is not guaranteed that a mobile phone will notify or even ask for permission from a user while an attacker can perform the following actions:

- Steal sensitive data such as restricted device numbers (e.g. UDID) which play a key role in the Mactans vulnerability [17]
- All data stored on the device
- Install malware using weaknesses in the connected host OS [24].

Note that in newer mobile phones restrictions have been implemented. Nevertheless, completely avoided are neither of those attacks.

c) Cold boot attacks: Not only the OS places built-in trust into USB, but also the UEFI/BIOS of the hosts system. How Wand and Stavrou point out by combining the already presented attacks of composite devices, it is possible to use an USB flash drive to also emulate a keyboard. This enables an intrusive device to alter the boot order by using an emulated keyboard and boot the OS from a rootkit saved on the device [32]. While *SecureBoot* enables another layer of protection, vulnerabilities in UEFIs have been discovered, such as BlackLotus [1].

2) Proof of concept: A Buffer Overflow (BO) is often reachable through not handling strings in a manner which guarantees that no data is written outside the designated buffer. If it does, arbitrary code execution is most likely a direct consequence, achieved by altering the program flow. Here, we take a close look into a reported Linux vulnerability from 2009 [10].

Field	Value	Description
bLength	14	Descriptor Length
bDescriptorType	3	String Descriptor
bString	”iPhone”	e.g. device name

Fig. 5. Typical String descriptor [9]

During the process of enumeration, multiple String Descriptors are sent to the host. They contain information such as the manufacturers name, Vendor-ID and the device's name. A typical structure of a string descriptor is shown in Fig. 5. The maximal length of the whole descriptor and therefore the highest value of bLength is set by: one byte for bLength field, one byte for bDescriptorType and originally 126 bytes (ASCII-encoded) for the bString field, making a total of 128 bytes. The string format had later been changed to UTF16-LE, which takes two bytes per character, increasing the maximum length to 252 bytes (126*2). So, if a driver allocates less than 252 bytes to save bString, a buffer overflow occurs and code execution in kernel mode is possible by e.g. using ROPgadgets. The code from a driver seen in Fig. 6 clearly only allocates 80 bytes of storage for the variable *name*, which saves the device name. With no firmware verification, it is easily possible to change this descriptor and therefore abuse the vulnerability. By ensuring that the USB firmware and thus all Descriptors are verified, this example could not have been discovered by an attacker since all strings are fixed. Of course, this does not implicate that no manufacturer would have used a device

```
struct snd_pcm {
    <cut>
    unsigned short dev_class;
    unsigned short dev_subclass;
    char id[64];
    char name[80];
    <cut>
```

Fig. 6. Source code from /linux-2.6.38/include/sound/pcm.h

name with more than 40 characters, therefore not completely preventing this attack.

D. USB device attacks

Inverting the attack vectors presented so far, attacks directed from the host USB devices can also indirectly put the host in danger. Especially air-gapped systems are endangered because an essential part is to not be able to communicate with the outside world and that data within the system stays in the system. USB undermines this principle by offering a quick way to transfer data to and from the machine with a simple USB flash drive or more complex attacks as we will see. To highlight the extent, Snowden used a flash drive to leak a substantial amount of classified documents [34]. Also, simply reading data from any flash drive from any computer is possible through the absence of proper standard-encryption mechanisms.

1) Lack of access control to the USB device: The host OS allows unrestricted access to a set of instructions to the provided driver of the USB system. Different security issues arise with not authenticating which applications communicate with which device. Every application can

- transmit data to a device
- find implementation errors in devices

While a control mechanism in the OS requires root privileges for e.g. creating a file on a flash drive [12], it is not necessary for an application to authenticate itself for solely transmitting data. This allows an unprivileged application to fuzz the USB device, enabling DOS attacks by blocking the communication channel with traffic and also flashing new firmware on devices like a webcam to change their intended behaviors. Brocker and Checkoway exploited this property to disable the indicator light of the MacBook Webcam though the camera is turned on [6].

2) Proof of concept: USBee is a method to turn any USB connector cable into a short-range RF transmitter [12]. Targeting systems which are not connected to the internet (air-gapped systems), Grui et al. can successfully transmit data from these systems to a RF receiver by sending encoded data camouflaged as outgoing file transfer to a USB drive. Through the encoding of the sent data via NRZI and the USB's clock speed, a specific and measurable frequency is generated when sending a sequence of 0-bits. This changes for a sequence of 1-bits and therefore allows a modulation of data on these two

sequences which one wants to send. An effective rate of 80 bytes/sec has been reached.

V. Possible security counter measurements

A. Missing communication channel encryption

To defend against the demonstrated USB sniffing attack Neugschwandtner propose UScamBle [19], a lightweight encryption solution which does not allow an attacker to listen to broadcasted data anymore. The key point is using the upwards communication channel for key exchange. It is unidirectional and does not have to encrypted. Because the attacker has access to an encryption oracle, namely the Host, UScramBle uses nonces when encrypting to prevent that random messages can be sent to the Host and later can be checked if the recorded traffic matches. Down streamed traffic is therefore encrypted with the use of AES-CTR. By using a `GetDescriptor` instruction, the host asks the device for a random key. The device then uses it to decrypt incoming messages while sending data unencrypted. The attack described only applies to USB version 3.0 or higher because broadcasting was disabled here for faster communication. However, most peripheral devices are not in need for a higher communication speed than USB 2.0. Through the given backwards combability this means this attack is still a major security issue. A minor issue for this solution is the well-known absence of enough entropy in embedded devices. Note that this solution only works for this scenario. E.g. a keylogger would receive downstream-, but also upstream data and therefore allowing a MitM attack.

B. Missing firmware authentication

Cronin et al. propose a data-driven approach to effectively verify whether a USB flash drive has been tampered with. Through collecting data on how much time it takes for verified USB device to reply to a specific operation, a baseline and an identifier for the verification progress is created. IN transfer with Bulk transfer type and an OUT transfer with the control transfer type represent respectively how the timing is measured with USB flash drives. The time spent is different for every USB device because of manufacturer differences and device-specific characteristics and thus enables precise recognition and validation of a known or unknown device model with greater than 99.5% accuracy [8].

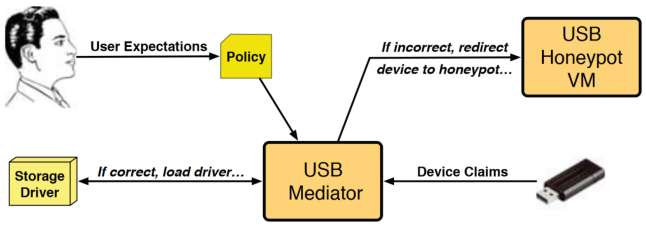

Fig. 7. GoodUSB functionality [27]

C. Plug and Play property and compound devices

BadUSB allows to quickly tamper with devices because drivers provide a set of instructions which normally only the user should access. To prevent devices from enumerating as arbitrary device, GoodUSB introduces a new security layer to the protocol. Tian et al. interpose a USB mediator between the OS and the host controller to enforce a policy on every USB device. The policy ensures the device operates within the user's expected functionality trough a graphical prompt. If the behavior is not aligned with the policy, the device will get flagged and gets redirected to a virtualized honeypot (HoneyUSB), see Figure 7. Here, monitoring USB traffic and analyze it for malicious behavior like keystrokes or IP packets would allow an IDS (Intruder Detection System) to evaluate if the device was correctly flagged. It is thus guaranteed that no drivers will be loaded before the user authorizes the device manually. GoodUSB's overhead is with 7 milliseconds during enumeration neglectable, and the graphical interface combined with the non-technical installation allow a wide implementation [27]. Other solutions like USBCheckIn work with a similar, but more basic principle [11].

D. Lack of access control to the USB interface on Host/OS

To prevent applications to arbitrarily communicate with USB devices and therefore enable attacks like USBee, it is necessary to restrict access to the USB interfaces. Throughout our research we did not come across over an existing solution for this specific problem. The research done by Scaife et al. proposes a software called USBFilter to mitigate attacks like BadUSB. While a big part of the solution is similar to GoodUSB's approach, the software also covers how only specific applications can access interfaces which they were assigned for. Note that GoodUSB is much more user friendly and to use USBFilter, one needs deep understanding of the USB protocol and the targeted USB device. It is thus not feasible to implement it in a broader scope for end-users to this point. Also, with this solution the implementation on the respective OS is more feasible than with any other solution.

E. No restrictions on physical layer

Because physical access is needed for every USB attack, regardless of whether the attacker himself deploys the device or social engineering leads to plugged in devices. To restrict access where USB ports are usually not needed, there exist USB port lockers which block devices from plugging into connectors [18] [4]. Additionally, data line blockers exist for charging a phone publicly. It is guaranteed that only the power lines of the interface are used and no data is transmitted to the connected hub. [23]

VI. Evaluation

We have shown how a wide spectrum of attacks look like and on what design choices they built on. Around these attacks we built a classification framework which highlights the most important oversights during the design of USB. This allows individuals and companies to make informed decisions

around USB, but also emphasizes the need of change in future versions. Though some attacks are more feasible than others, all attack vectors are currently open. To mitigate these vectors, we showed what solutions latest research provides and what user-friendly features of USB are suffering through the limitations these defenses bring with themselves.

During our research we found USB based attacks where a clear classification based on design decisions of USB is not possible:

a) It is possible for a developer to deploy a self-written USB driver to the Microsoft Website. Though there are requirements, one can bypass them by stealing an already present certificate and credentials and pose as a company which has been proven. Only the signed binary code is checked from Microsoft and therefore it is possible to deploy malicious code in it. This would be run on every Windows machine if a USB device requests this driver. This was e.g. used in the worm Stuxnet, likely developed by government agencies.

b) COTTONMOUTH is a malicious USB cable designed by the NSA which disguises a built-in RF antenna inside and can broadcast data just like USBee. It differs in the way that the antenna can explicitly be the target endpoint and no data has to written to the USB device USBee targets. However, the proposed solution for controlling access to USB interfaces from the host would prevent such an attack.

c) Malware stored on hidden parts of USB devices prevents anti-virus software from scanning a flash drive completely. The malware Fanny used a magic value for finding its own code in sections, that were marked as reserved or defect.

d) USB based attacks that involve breaking out of a Virtual Machine (VM) cannot be classified within a separate context. Vulnerabilites can lie in the implementation of the VM and the host OS [6].

VII. Discussion

Why did we choose Windows for specific examples?
Throughout this paper we analyzed attacks on both Windows and Linux operating systems. While Linux is mostly used by user which can assess the risk of USB based attacks better, Windows dominates the consumer market and often is used by unsuspecting consumers. Some attacks like Juice Jacking were directed at iOS, but are outdated. We have therefore made a few selections in favor of Windows over Linux and iOS.

Is USB device behavior affected by a proposed solution?
GoodUSB and other solutions were tested against various devices like webcams and headsets. No malfunction has occurred while only enabling one function. [27] and overhead was always neglectable for users.

Can a USBKiller attack be prevented from software?
USBKiller is not affected by the PnP property. For the USB system to detect a device, power must be transmitted thus charging the malicious device so even devices without an internal battery can damage a system.

Every attack method uses properties shared with other proposed classes. This can be explained by the fact that those properties are fundamental for the communication to work. It is classified by the main property supporting this attack and thus having the highest impact.

VIII. Conclusion / Future Work

USB ports nowadays are practically on every device while bringing a highly versatile interface to the user. With its common use, it is strongly unsettling how fast and easy USB based attacks can be implemented. This paper presents a classification of USB-based attacks and by providing real-world case studies, it highlights the severity, impact, and feasibility of these vulnerabilities. To increase security within this realm, different countermeasures have been proposed and user awareness has been created. Future research should focus on designing and establishing an industry-wide standard to ensure flaws in the covered design decisions do not affect upcoming USB versions. While security improvements in the USB protocol are overdue, the biggest threat is still the human factor. "If you let someone connect a USB thumb drive or charge a phone on your computer you essentially trust them to type commands on your computer" [7].

References

[1] BlackLotus UEFI bootkit: Myth confirmed.

[2] Brian Anderson and Barbara Anderson. *Seven Deadliest USB Attacks*. Syngress Publishing.

[3] Don Anderson. *USB system architecture*. PC system architecture series. Addison-Wesley Developers Press.

[4] Olga Angelopoulou, Seyedali Pourmoafi, Andrew Jones, and Gaurav Sharma. Killing your device via your USB port.

[5] Jan Axelson. *USB Complete The Developer's Guide*. y Lakeview Research LLC, 5310 Chinook Ln., Madison WI 53704, 5 edition.

[6] Matthew Brocker and Stephen Checkoway. iiSeeYou: Disabling the MacBook webcam indicator LED.

[7] Lucian Constantin. Thumb drives can be reprogrammed to infect computers.

[8] Patrick Cronin, Xing Gao, Haining Wang, and Chase Cotton. Timeprint: Authenticating USB flash drives with novel timing fingerprints. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1002–1017. ISSN: 2375-1207.

[9] Andy Davis. USB – undermining security barriers.

[10] Any Davis. Lessons learned from 50 bugs : Common USB driver vulnerabilities. NCC Group Publication.

[11] Federico Griscioli, Maurizio Pizzonia, and Marco Sacchetti. USBCheckIn: Preventing BadUSB attacks by forcing human-device interaction. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 493–496.

[12] 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.

[13] Intel, Hewlett-Packard, Microsoft, and NEC. USB type-c specification r2.0.

[14] KeyGrabber. Hardware keylogger - KeyGrabber USB.

[15] Kingston. IronKey - kingston technology.

[16] Yuvraj Kumar. Juice jacking - the USB charger scam.

[17] Billy Lau, Yeongjin Jang, Chengyu Song, Tielei Wang, Pak Ho Chung, and Paul Royal. Mactans: Injecting malware into iOS devices via malicious chargers.

[18] LINDY. USB typ a port locker.

[19] 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. ACM.

[20] Nir Nissim, Ran Yahalom, and Yuval Elovici. USB-based attacks. 70:675–688.
[21] Karsten Nohl, Sascha Krißler, and Jakob Lell. BadUSB — on accessories that turn evil.
[22] David Oswald, Bastian Richter, and Christof Paar. *Side-Channel Attacks on the Yubikey 2 One-Time Password Generator pp 204-222*, volume 8145 of *International Workshop on Recent Advances in Intrusion Detecion*. Springer Berlin Heidelberg.
[23] PRIVISE. PRIVISE: Privacy gadgets.
[24] Debabrata Singh, Anil Kumar Biswal, Debabrata Samanta, Dilbag Singh, and Heung-No Lee. Juice jacking: Security issues and improvements in USB technology. 14(2):939. Number: 2.
[25] Yang Su, Damith Ranasinghe, Daniel Genkin, and Yuval Yarom. USB snooping made easy: Crosstalk leakage attacks on USB hubs.
[26] Chengzhi Sun, Jiyu Lu, and Yunqing Liu. Analysis and prevention of information security of USB. In *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, pages 25–32.
[27] Dave Jing Tian, Adam Bates, and Kevin Butler. Defending against malicious USB firmware with GoodUSB. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 261–270. ACM.
[28] Jing Tian, Nolen Scaife, Deepak Kumar, Michael Bailey, Adam Bates, and Kevin Butler. SoK: "plug & pray" today – understanding USB insecurity in versions 1 through c. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 1032–1047. ISSN: 2375-1207.
[29] Matthew Tischer, Zakir Durumeric, Sam Foster, Sunny Duan, Alec Mori, Elie Bursztein, and Michael Bailey. Users really do plug in USB drives they find. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 306–319. IEEE.
[30] USBKill. USB kill devices for pentesting & law-enforcement.
[31] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards.
[32] Zhaohui Wang and Angelos Stavrou. Exploiting smart-phone USB connectivity for fun and profit. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 357–366. ACM.
[33] Wikipedia. Fuzzing. Page Version ID: 225194301.
[34] Kim Zetter. Snowden smuggled documents from NSA on a thumb drive. Section: tags.