Classification of USB based attacks

~~170-250 words~~ ***Abstract*—This research paper presents a comprehensive classification of USB-based attacks, focusing on the underlying security flaws within the USB specification.**

By understanding these weaknesses, individuals and cooperations can better assess the risks associated with USB usage and implement adequate security measurements. Furthermore, this paper proposes security countermeasurements and evaluates them, ~~offering~~ to make informed decisions about them.

***Index Terms*—~~component, formatting, style, styling, insert~~**

allowing?

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) different 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 a device in an out while the system is running (hot pluggable) and the integrated power supply have given the user a easy-to-use interface. Manufacturers benefit from cost-efficient production of products because there is no intelligence in the USB device itself necessary; the host takes over communication and computation. Finally, Universal Serial Bus (USB) offers one interface for almost every kind of device, therefore it finds application from mobile storage to peripheral devices to USB-powered fans.

does the device not still require 'intelligence' to interface keystokes/ mouse movement/flash... to the usb bus?

As the history of USB starts in 1996 with the introduction of USB 1.0, the protocol was firstly supported by Microsoft Windows 95. Following ~~to~~ 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 meets. This met the requirements for most USB devices while maintaining backwards compability and thus became the most widely adopted version of USB. In 2008, the USB Implementers 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 10Gbps, referred to 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 [?]. 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 changend over the years. Fundamental components of the USB protocol like *trust-by-default*, non-encrypted communication, *autorun* and more represent a big attack surface to this interface. Adding to that, according to a study done by Tischer et al. [?] 45%–98% of "lost" USB flash drives get connected to a PC, the first in this study 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 lead to an overall insecure system as soon as 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 OS and show the systematic behind each class on the basis of one representative. Finally, researched solutions to these inherent flaws of USB are presented, explained and evaluated based on the proposed classes.

operating system (OS)

A. Summary

Is the section heading needed here, as it is the only subsection?

This paper presents the following contributions:

- Discussing the non-existance of adiquet 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 ~~to~~ in that regard
- Explaining technical details used for attacks in this realm
- Proposing researched security concepts and evaluate 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 interest.

A. Differences in communication in USB versions

From USB 1.x, USB 2.0 enhanced the bus protocol with reduced transaction overhead while still using a shared bus architeture similar to USB 1.x and being backward compatible to it. As of USB 3.0 a new bus architecture is introduced with seperate transmit and receive channels and allowing traffic to be directed to the specified device. Enhanced SuperSpeed refers to devices supporting SuperSpeed and SuperSpeedPlus, therefore version USB 3.1 Gen. 1 and higher. With USB 3.0 and higher 2 seperate data lines are added.

USB version differences			
Version	Speed name	max. data rate	max. power
USB 1.0	Low Speed	1,5Mbit/s	0,5W
USB 1.0	Full Speed	12Mbit/s	0,5W
USB 2.0	Hi-Speed	480Mbit/s	2,5W
USB 3.0/3.2 Gen 1	SuperSpeed	5Gbit/s	
USB 3.1/USB 3.2 Gen 2	SuperSpeedPlus	10Gbit/s	
USB 3.2 Gen 2x2	SuperSpeed20	20Gbit/s	4,5W
USB 4.0	USB 40	40Gbit/s	4,5W

Fig. 1. Different protocol speeds and supported power outputs

B. Function, Device

One device can have many functions, e.g. one USB connection with a printer and scanner funcionality. Each function has its own endpoint.

C. Root hub, Host controller

The root hub operates as source of all USB ports and works 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 responisble for monitoring device status and configuration aswell 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. 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 to ensure that every device can send and receive data and does not starve.

D. Transfers

To interact with a device, the USB host always has to iniciate the transfer (except for remote wakeup signaling). The USB device then responds to the data sent. Data in a USB device directed to a function 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 to the device or an OUT signal for requesting data. It respectively responds with either a status signal or the requested data.

E. Transfer Types

There are 4 different ways to communicate on the protocol:

- Bulk
- Control
- Interrupt
- Isochronous

Bulk is mostly used in printers and flash drives where there is no need for fast reaction time. It is fastest if the bus is idle, and waits, if not. Control is a type which is mandatory to be supported for every device. In enumeration this is the standard protocol. If a guranteed maxium 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. Isochronous transfer also ensures a maximum guarranteed latency but differs ~~itself~~ by limiting the response to a defined number of bytes in a specific interval. Primarily utilized in the area of streaming audio or video, it lacks support for error correction.

F. Enumeration

On attaching, the host system recognises 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 is requested, including

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

For devices with more than one function *(composite devices)* different configuration decriptors 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 set transfer type to an interface. Note that speeds from USB 2.0 or lower are negotiated before the GetDescriptor request by using the power system. With a SetConfiguration 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.

G. Bus speed decision

A USB device employs the highest mutually supported USB speed when connected to the nearest hub. The hub communicates with its parent device at the highest common USB speed, even if the connected devices operate at a lower speed.

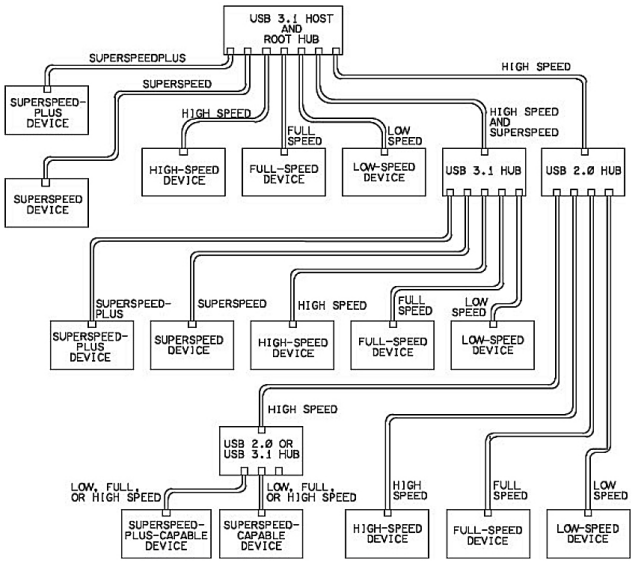

Fig. 2. Progression of USB speeds across differnt hubs and devices

Fig. 3. USB Type-A male

H. Physical construction

USB Type-A male pinouts consist of a set of four pins that define the electrical connections. Pin 1, denoted as VBUS or +5V, 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+), which during normal operation transmits the inverted data of Pin 2. Pin 4 functions as the ground (GND) reference. To interact on the physical layer of the USB protocol, a USB controller is used. For instruction purposes the components and processes here listed are only referring to the USB 2.0 specifications. 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 gets 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 Bitstuffing. This adds a 0 bit after six consecutive 1 bit's. 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.

I. Drivers (Autorun?)

III. Related Work

Throughout this paper we will use examples of USB based attacks of 29 typical attacks introduced by Nissim [7]. Tian et al. presented 2018 a classification and enlisting of attacks from which we will find inherent design flaws in the USB protocol [10]. At the end, we provide an overview over existing countermeasurements covering our classification proposed by numerous papers implementing the attack representative of a reprentative. 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 weakpoints in the specification are.

IV. Exploiting fundamental design choices in the USB protocol

We start by grouping USB based attacks into 3 different categories:

1) Modified Hardware: To attack in this area, a specific hardware has to be implemented and used. Regular USB devices do not offer enough capacibilities to execute them.
   Issues here: missing encryption, physical access bietet Möglichkeit zur Zerstörung
2) Host-Software attacks: Attacks are directed from a USB device to the host and vice versa, directed to specific components. Drivers from the OS or the USB device itself can be targeted.
3) Unexpected funcionality: Through changes in the interfaces a USB device offers 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 very broad.

After framing USB's flaws we categorize and discuss different attacks within a group. We then propose possible security countermeasurements for each individual group.

A. Modified Hardware

Over the years a wide range of hardware based attacks on the hardware side of the USB protocol has been discovered. Major 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 aswell as less overhead in the communication. We will show in the proposed solutions that the effect of the latter can be neglected.

1) Missing communication channel encryption: Besides 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 up the possibility of intercepting data using a MitM attack, but also allows other side-channels of USB like eletromagnetic fields or transfer types being used to be read. [?] [?].

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 if powered up or not [?].

3) Proof of concept: To illustrate a possible attack vector, we refer to Neugschwandtner et al. [6]. 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. 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.

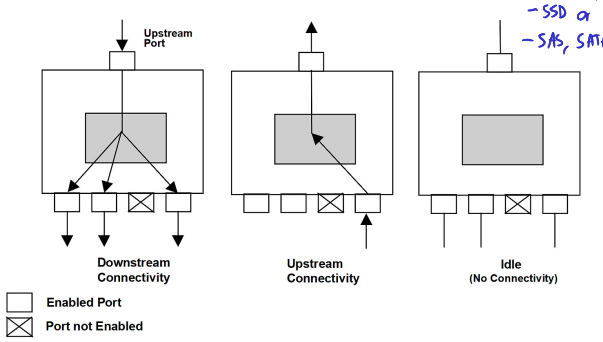

Fig. 4. Hub connectivity [9] Section 11.1.2.1

Through the process of how the bus speed and thus the version of the USB protocol gets chosen, it is possible to eavedrop downstream traffic (OUT packets) from the point a USB 2.0 hub gets used. Regardless if hubs or devices in ranks further away from the host use versions higher than USB 2.0, 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 [9] (USB snooping made easy: Crosstalk). Generally, 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 behaviour 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 [6] (CITED BUT DETAILS ADDED). Similar eavedropping attacks have been done by measuring differences in voltage in an USB system [?]. A much easier method to implement for sniffing traffic is using a traffic logger which is plugged in between root host and the target device. While logging e.g. keystrokes with a device such as [?] 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 preferabely not often unplugged. We conclude that the foundation of every physical wiretap attack 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 [?]. This mechanism is undermined in the scope of this attack, even on a maleware-free OS.

Another attack in this realm present USB Killers [?]. A capacitor is charged by the power supply of the host, providing electricity as shown in [?]. By discharging a high voltage pulse through the USB interface into the USB root hub, it damages the underlying electronical system. The underlying security flaw here is the abscence of protection measurements, such as voltage regulation circuts. While a device can be seriously damaged this way, [?] shows that data stored on hard disks (SSD or SATA) remains intact after such an attack. Thererfore, 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 [?]

B. Unexpected software functionality

Due to 3 different USB design decisions making it possible to emulate functions, which an unsuspecting user would not expect, a big class of attack vectors is created. Many known attacks in this section are generally known under the category of BadUSB attacks, such as Rubber Ducky, Evilduino, USB-driveby and more [7] [?]. Note that though mostly USB drives or specfic hardware components are used within BadUSB, an attacker can also deploy such functionalities in e.g. a mouse [?]. While the from Windows provided autorun caused vulnerabilites by itself, running malicious code automatically as soon as connected in the past was fixed in [?]. TO BE ADDED

1) Missing firmware authentification: The inability to prove whether a device is from a trusted source, (aka a official vendor) or not opens doors for anybody to create a 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, not properly managed this is a key factor of Host-Software attacks. By not asking a user for validation of a freshly added device, it allows also installations while a computer is locked, e.g. by a screen saver.

3) Composite devices: The concept of composite devices allow devices to provide a set of funcionality within one device. Commonly uesd in headsets or webcams, one USB connection can be suffient 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 [7] (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 controller. With two different device descriptors, the attached device enlists at the host in the enumeration process as 1) a USB thumb drive

Does the USB device act as a DHCP server or does it act as a NIC which has a DHCP server connected?

and 2) as an emulated DHCP (Dynamic Host Configuration Protocol) server. By setting the InterfaceClass field of one of the devices 2 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 simultaniosely as a normal USB flash drive. By only adding a DNS (Domain Name System) server entry and not setting a Default Gateway, the internet connection up to now stays as is and only the DNS server is changed. 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." [?]. 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 a MitM attack. By controlling the 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 [?]. [?]

How exactly? Additionally added?

How? Changing DNS server to redirect traffic to own server won't work with HTTPS/TLS.

does it not work on a stock phone?

C. Host/OS attacks

While software attacks showed up to this point have used legitimate USB device behaviour, we now look into possible vulnerabilities arising through built-in trust in communication points with the USB host. The goal is to run arbitrary code on the host. We achieve this by forcing non-standard behaviour and the ability to access confidential data from USB components to bypass restrictions and protection measurements which are made by the host.

1) Inherent trust of the host system on plugged in devices: This property enables a large spectrum of USB attacks. To better understand the dynamics and effects, 3 examples are elaborated. Like the PnP property, it increases usability by making a device accessable in a fast manner.

1) Driver fuzzing: Fuzzing is the process of sending a large volume of random input, valid or invalid, to find differences in how the program reacts based on this input [?]. With this and other techniques many flaws within USB drivers have been found over the years [2]. 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. Rather the implicit trust placed on the connected device is the root cause for an attack vector like this. Although we can realize an attack, 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 code for security vulnerabilities. By explaining the functionality of this attack in a proof of concept, we better understand where software related security issues, away from the actual USB protocol, start to appear.
2) Juice Jacking: Relating to the work of Lau et al. [1] in a wide spread vulnerability in IOS devices brought attention tho the realm of Juice Jacking. As we see more and more publicly available USB ports, it is critical to understand what kind of trust you place on the e.g. wallcharger 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 such as the UDID playing a key role in the Mactans vulnerability [1] or simply data stored on the device
   - Install malware using weaknesses in the connected host OS [8].
3) 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 attack of composite devices, it is possible to use an USB flash drive to also emulate a keyboard. This enables an intrusive device to boot the OS from a rootkit saved on the device and altering the boot order by using the emulated keyboard [?].

Unique Device Identifier (UDID)

don't phones ack on connection? (except in that 3 case)

what about secure Boot?

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. ~~We~~ Here, we take a close look into a reported Linux vulnerability from 2009 [5].

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

Fig. 5. Typical String descriptor [4]

During the process of enumeration multiple String Descriptors are getting sent to the host. They contain information such as the manufacturer and the device's name. A typical structure of a String descriptor is shown in Fig. 5. The maximal length of a String descriptor is set by: one byte for bLength field, one byte for bDescriptorType and originally 126 bytes (ASCII) characters for the bString field, making a total of 128 bytes. The String format had been changed to UTF16-LE, which takes 2 bytes per character, increasing the maximum length to 252 bytes. By ensuring that the USB firmware and thus all Descriptors are verified, this example could not have been discovered by fuzzing techniques since all Strings are fixed. Of course, this

where?

The example does not use fuzzing.

```
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

does not implicate that no manufacturer would have used a device name with more than 40 characters, therefore not completely preventing this attack.

The variable *name*, which saves the device name, clearly only allocates 80 bytes of storage, whereas seen above a maximum of 252 bytes is possible. Code execution in kernel mode is therefore made possible e.g. by correctly setting the return address to a ROPgadget.

D. USB device attacks

Inverting the attack vectors presented so far, attacks directed from the host USB devices can also indirectely put the host in danger.

1) Lack of access control to the USB interface: 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 interfaces:

2) Proof of concept: USBee is a method to turn any USB connector cable into a short-range RF transmitter [**?**]. Targeting systems which are not connected to the internet (air-gapped systems), Grui et al. can successfully transmit data from them to a RF receiver by sending data camouflaged as outgoing file transfer to a USB drive.

V. Possible Security Countermeasurements

A. Missing communication channel encryption

B. Missing firmware athentification

C. Plug and Play property

D. Lack of access control to the USB interface

VI. Evaluation

VII. Discussion

USBKiller is not affected by the PnP property. For the USB system to detect a device, power has to be transmitted thus charging the malicous device.

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 has the highest impact.

Modified hardware which communicates with the host can not be classified by the proposed framework. One example is COTTONMOUTH, a malicious USB cable designed by the NSA, which has a built-in RF antenna and can transmit data just like USBee. It differs in the way that the antenna can explicitaley be the target endpoint.

Developing a driver: This attack may not be the most feasible, but it highlights how a system is affected when

We do not evaluate attacks happening between host and applications, therefore not covering VM based attacks like other research does.

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 attacks on the protocol can be implemented. This paper presented a classification of USB-based attacks and by providing real-world case studies, it highlighted the severity, impact and feasibility of these vulnerabilites. 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, on a physical level 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" [**?**]

References

[1] Mactans: Injecting malware into iOS devices via malicious chargers.

[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] Andy Davis. USB – undermining security barriers.

[5] Any Davis. Lessons learned from 50 bugs : Common USB driver vulnerabilities.

[6] 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.

[7] Nir Nissim, Ran Yahalom, and Yuval Elovici. USB-based attacks. 70:675–688.

[8] 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.

[9] Yang Su, Damith Ranasinghe, Daniel Genkin, and Yuval Yarom. USB snooping made easy: Crosstalk leakage attacks on USB hubs.

[10] 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.