SoK: "Plug & Pray" Today – Understanding USB Insecurity in Versions 1 through C

***Abstract*—USB-based attacks have increased in complexity in recent years. Modern attacks now incorporate a wide range of attack vectors, from social engineering to signal injection. To address these security challenges, the security community has responded with a growing set of fragmented defenses. In this work, we survey and categorize USB attacks and defenses, unifying observations from peer-reviewed research and industry solutions. Our systematization extracts offensive and defensive primitives that operate across layers of communication within the USB ecosystem. Based on our taxonomy, we discover that USB attacks often abuse the Trust-by-Default nature in the ecosystem, and transcend different layers within a software stack; none of the existing defenses provides a complete solution, and solutions expanding multiple layers are most effective. We then turn to the first formal verification of the recently released *USB Type-C Authentication* specification, and uncover fundamental flaws in the specification's design. We further evaluate the spec using findings from our systematization, and find that while the spec has successfully pinpointed an urgent need to solve the USB security problem, those flaws render its goals in vain. We conclude by outlining future research directions to ensure a safer computing experience with USB.**

I. Introduction

Since the early 2000s, USB devices have increased in popularity, with new features paving the way for widespread adoption in nearly every computing device [35]. The newest iteration of USB, USB Type-C, has strong support from popular vendors, for example, Type-C is now the exclusive means of peripheral interaction with Apple MacBooks and new Google smartphones [76].

Unfortunately, USB innovation has largely left security as an afterthought. New specifications rarely mention security, and until recently, USB designers placed the onus of security onto the consumers and vendors of USB devices [117]. As a result, USB devices are often a ripe target for attackers. We begin this work with a systematic analysis of the attacks present in the USB ecosystem. We find that many USB attacks appear at varying communication layers, ranging from the human layer (social engineering) down to the physical layer (signal injection). In addition, all attacks abuse the "trust-by-default" nature of the USB ecosystem.

In spite of the evolving USB threat landscape, defenses against such attacks are fragmented and not widely adopted. In systematizing the defenses present in the USB ecosystem, we find that most defenses often focus on protecting a single layer, which proves ineffective against a suite of attacks that appear at many communication layers. In addition, misaligned goals between industry and academia further fragment the defense space. Commercial solutions focus on the prevention of data loss and anti-malware without regard for emerging attack vectors, while research prototypes vary in are hamstrung by the lack of built-in security building blocks in the existing USB specifications. As a result, research solutions often rely on new host and peripheral architectures that are unlikely to be incorporated into commercial systems.

After years of USB insecurity, the USB Implementers Forum (USB-IF) incorporated security features into the most recent specification, USB Type-C. The new specification enables Type-C authentication, which is intended to provide a way to authenticate a USB device before interacting with it. However, it is unclear whether authentication is sufficient to defend against all existing attacks, and what can be done for legacy devices which do not support the new spec. To investigate these questions, We formally verify the USB Type-C authentication protocol. Though the spirit of the specification highlights long-awaited attention to security by USB designers, we find multiple attacks that can break the security guarantees provided by the specification. We argue that had the USB Type-C designers learned from the attacks and defenses of the past, many specification flaws could have been mitigated. We further leverage our systematization to pinpoint what security issues the new protocol addresses, and more importantly, where it still fails.

We conclude with a discussion of future directions for USB security, leveraging our taxonomy and systematization to focus attention on the problems that remain. We hope our results prove useful to the security community as we work towards a safer USB computing ecosystem.

II. Background

We first outline the evolution of the USB specification and highlight key features that inform the present state of USB security.

A. The Evolution of USB

Introduced in 1996, USB 1.0 [31] was designed to replace disparate peripheral connecting interfaces and reduce the complexity of both hardware design and software configuration. USB 1.x [31], [32] features a polled bus, meaning that the USB host controller initiates all data transfers. It provides two data transfer rates, which are known as Low Speed (1.5 Mbit/s) and Full Speed (12 Mbit/s). USB 1.x additionally provides a limited amount of power of over the cable for "bus-powered" devices. The term "security" does not appear at all in the USB 1.x specification; the closest related topic is error detection in the cable during transmission.

In 2000, the USB 2.0 protocol specification was released. USB 2.0 provided increased peripheral support and a High Speed (480 Mbit/s) data transfer rate. Peripheral support was expanded to include digital cameras, video cards, CD writers, and network adapters (in particular, 802.11 and Bluetooth). USB 2.0 also paved the way for the popularity of "flash drives"—portable devices that enabled physically transferring data on the go. Like the 1.x specifications, the security of USB devices is not stressed in the 650 page document. The lone exception is the introduction of a new peripheral class called Content Security [46], which attempts to provide limited support for securing sensitive content, for example, readings from fingerprint scanners.

USB 3.0 [55] was published in 2008, and offers a SuperSpeed (5 Gbit/s) data transfer rate. Like 2.0 before it, USB 3.0 offered expanded support for new classes of peripherals, such as USB Vision [9] for controlling cameras and external USB-based graphics processing [36]. USB 3.0 also replaced the downstream traffic broadcast mechanism used with a unicast protocol, enabling internal routing within hubs. The 2013 release of USB 3.1 [56] brought about SuperSpeed+ (10 Gbit/s) as well as an updated USB Power Delivery (PD) specification [47]. This specification, which supports up to a 100W power supply over USB, paved the way for laptop charging via USB. Unfortunately, security continued to remain absent from the 3.x specification. USB Type-C [57] was introduced as a part of USB 3.1 as a new connector type, unifying PD, USB 3.x, Thunderbolt, DisplayPort and HDMI using a 24-pin connector/cable. In 2017, USB 3.2 [14] was released, and doubles the data transfer rate of previous generations (20 Gbit/s).

Throughout the evolution of the USB protocol, security was rarely given consideration. As recently as 2014, the USB Implementors Forum (USB-IF) explicitly stated that security falls outside the scope of the USB specification. In an official statement [117], the USB-IF asserted that security is not a legitimate concern because "In order for a USB device to be corrupted, the offender would need to have physical access to the USB device." They continue by placing the onus of security onto onto both the consumers of USB products and the original equipment manufacturers (OEMs).

They state:

1) "OEMs decide whether or not to implement these [security] capabilities in their products."
2) "Consumers should always ensure their devices are from a trusted source and that only trusted sources interact with their devices."

However, the USB-IF could not ignore security for much longer. In response to the threat of rogue power chargers and cables enabled by the USB Type-C specification, the USB 3.0 Promoter Group and the USB-IF introduced the USB Type-C Authentication specification [116] to Type-C products in 2016.

B. USB Protocol

The true flexibility of USB comes from *composite devices*, which can contain multiple configurations and interfaces, each of which is a standalone entity. For instance, a USB headset may contain one configuration, which contains four interfaces, including a keyboard (for volume control), a microphone, and two speakers. An example of a two-configuration USB device is shown in Figure 1. Two mechanisms are necessary to accomplish composite devices, one to define different kinds of peripherals, and another is to connect to them.

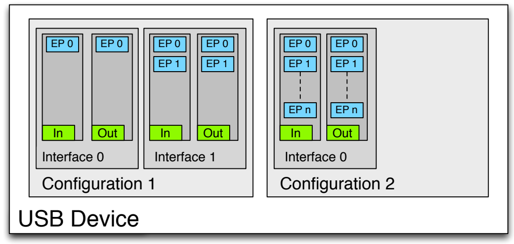

Figure 1: A USB device containing two configurations. Configuration 1 contains two interfaces, and configuration 2 contains one interface. Each interface supports two unidirectional communication channels (In/Out) with the host machine. Each channel may contain more than one endpoint (EP), which is the sink of the communication.

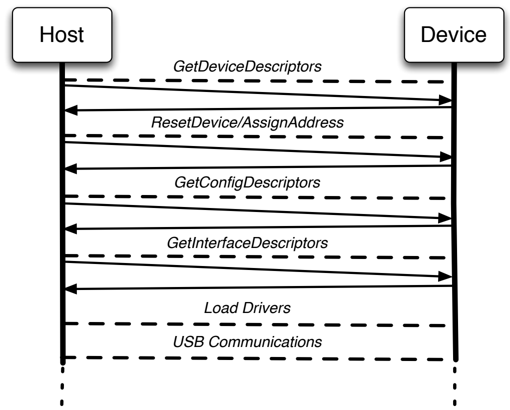

Figure 2: USB Enumeration Procedure.

1) Common Class Specifications: Beginning in USB 1.0, the notion of Common Class Specifications [34], [111] is introduced to codify different kinds of peripherals. A USB class is a grouping of one or more interfaces that combine to provide a more complex functionality. Examples of classes that feature a single interface include the Human Interface Device (HID) class that enables the USB host controller to interact with keyboards and mice, and the USB Mass Storage Class [119], [118] that defines how to transfer data between the host and storage devices. A composite device can then combine different classes to create a useful product, such as a USB headset leveraging both the HID class and Audio class. As we will see, the notion of designing USB peripherals through a composition of multiple functionalities continues to bear on the state of USB security today.

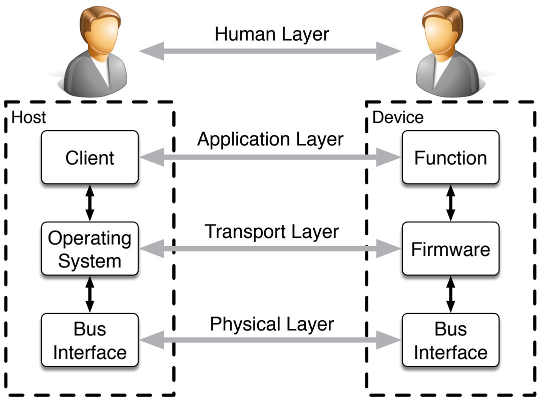

Figure 3: USB vulnerabilities can be classified by the abstracted communications layer at which they operated. A successful attack involves violating an design assumption or implementation error at a given layer.

2) Device Enumeration: After a device is plugged into the host machine, the USB host controller detects its presence and speed by checking the voltage change on data pins. Enumeration then begins as shown in Figure 2 with the `GetDeviceDescriptors` command, which asks the device for its identifying information including manufacturer, Vendor ID (VID), Product ID (PID), and serial number. The host controller then resets the device and assigns an address to the device for future communication. A `GetConfigDescriptors` request then obtains all configurations available within the device. A USB device can have one or more configurations, though only one may be active at a time. Each configuration then can include one or more interfaces, which are obtained with the `GetInterfaceDescriptors` request and represent the essential function entities served by different drivers within the operating system. After `GetInterfaceDescriptors` is done, drivers are loaded on behalf of the device and class-specific subsets of the USB protocol (e.g., HID, Storage) begin.

III. Understanding USB Attack Vectors

In this section, we explore the current attacks against USB. Given the myriad of work in this space, we first classify existing work via abstract communication *layers*. As seen in Figure 3, our layers represent the various entities involved across both the host and peripherals. At the highest level, the human layer involves actions and communications between human stakeholders. The application level layer represents user-level programs on the host and capabilities on the device. The transport layer encompasses both device firmware and host operating systems containing the USB stack. Finally, the physical layer represents the communication over the USB bus.

By grouping research into layers, we can more easily identify commonalities in approaches and derive sub-groupings, called *primitives*. In the case of attacks, these primitives encompass both the mechanism (i.e., how the attack is accomplished) and the outcome (e.g., forgery, eavesdropping, or denial of service). In the case of defenses, discussed in Section IV, these primitives likewise encompass mechanism, but instead highlight security guarantees (e.g., integrity, confidentiality, or availability).

A. Abuse of Human Layer

Abuse at the human layer involves social engineering or human error, as leveraged by outsiders as well as privileged members in an organization.

1) Outsider Threats: All USB attacks rely on the plugging in of a peripheral in order to damage a host or use, leading security practitioners to warn of the dangers of inserting suspicious devices into computers. *Social engineering* frequently involves tricking a user into plugging an untrusted device into their machine and interacting with its contents; in practice, this is not a challenging task. Stasiukonis reports that in a 2006 penetration test, compromise of the organization was made easy as 75% of USB drives scattered near the workplace had been plugged into company computers within three days [107]. The US Department of Homeland Security [89] replicated this result in a similar experiment where 60% of drives dropped found their way onto a government computer; this number increased to 90% when drivers were branded with a government logo, suggesting that users' low bar for electronic trust can be manipulated by attackers.

Wagenaar et al.'s 2011 "USB Baiting" experiment [122], also demonstrated that users plugged in USB drives and explored their reasons for doing so. Though one would expect general security awareness to increase over the years, recent work demonstrates empirically that users are *still* plugging in the USB drives they find [115], [61]. Extending other experiments, Tischer et al. planted appearance-modified drives to instigate different human motivation, such as altruism or self-interest. The researchers found 98% of drives were picked up from the drop site and that files on 45% of drives were actually opened. The ease of executing such attacks make USB-based social engineering attacks both realistic and dangerous.

2) Insider Threats: The ease and rapidly declining cost of USB drives enables both companies and consumers to leverage USB devices to store and transfer sensitive data. USB drives, like any physical device, can be damaged, or worse, lost, due to human error. Such mishandling can often lead to detrimental results. In 2011, Ponemon Institute released a study that documented 400 different companies; they found that these companies have lost >2.5 million dollars per company because of misplaced USB drives [75]. Later in 2011, an Australian defense aid lost top secret documents stored on a USB drive in transit through Kuwait [37]. Humans are error prone, and even honest parties can make mistakes that can heavily cost companies and even countries.

USB devices such as portable media also enable an easy avenue for inside attackers to leak sensitive information. In 2010, Private Chelsea Manning was arrested in violation of the Espionage Act, for leaking private U.S. army footage to

Wikileaks [69]. Manning used portable storage, for which exit restrictions were lax, in order to breach the air-gap in the army's network and leak information to the outside world. Edward Snowden used similar techniques to siphon top-secret NSA data from his Hawaii base [10]. These are only a few cases that we as a security community know of—it is possible and highly probable that USB storage has been used to conduct similar attacks in many different scenarios.

The ubiquity and portability of USB devices are both a challenge and an opportunity. On the one hand, their ease of use greatly aid consumers and companies in day to day tasks. On the other, USB devices are the de facto method of bypassing technical and personal security precautions and can lead to large, detrimental effects to organizations.

B. Abuse of Application Layer

Application layer attacks involve user-space processes on the host and their interactions with the functionality of a device. Attacks in this layer typically fall in two categories: code injection attacks, where the attacker injects malicious code into the host, and data exfiltration attacks, in which the device exfiltrates data from the host.

1) Code Injection: USB storage has been used by several high-profile malware attacks. Stuxnet [44], [30] allegedly attacked nuclear centrifuge equipment in an airgap environment after infection via USB storage drives. Duqu [109] uses a user-mode rootkit to hide malicious files on the USB storage device. Conficker [104], [45], [92] and Flame [127], [128], [123] used zero-day exploits and malicious `autorun.inf` files to automatically execute the malware when a storage device was connected to the host. Although the auto-run feature was restricted after it became one of the top threats for the Windows platform [78], similar functionality remains available due to bugs in the operating system [71].

2) Data Exfiltration: Since the USB device often does not authenticate the communicating application on the host, the device may send or receive sensitive data from an unintended application. This is particularly problematic for sensory devices that can be used to perform surveillance on an unsuspecting user. For example, webcam access has been leveraged by both government agencies [85] and malware [1], [98] to obtain information about the computer's user and environment. In the case of malware, the attackers can then demand a ransom payment from the user. Web pages may request that a vulnerable browser enable the microphone without the user's permission, allowing the site to capture audio from the system [110]. Portnoff et al. found that less than half of people noticed that their webcam indicator light illuminated during computer-based tasks [93]. Even worse, USBee [51] permits the exfiltration of data from the host system by turning any USB device connected with the machine into a RF transmitter. Similarly, an exploitation on Linux resource manager [29] allows arbitrary users to bypass system restrictions and access any USB devices on the system.

C. Abuse of Transport Layer

Attacks on the USB transport layer fall into two general categories: those that perform privilege escalation through additional interfaces and those that send maliciously crafted packets/messages to compromise the host operating system.

1) Protocol Masquerading: These devices provide additional, obscured interfaces to the host operating system leveraging the host's full trust of any connected device. When a device such as the Rubber Ducky [52], [53] or USBdriveby [66] connects to the host system, all of its interfaces – some of which are not known to the user – are enumerated. Such functionality can be hidden from the user by embedding additional functionality into an otherwise innocuous device (e.g., a network adapter in an audio headset). TURNIPSCHOOL [5] is a modified USB cable that contains an RF transmitter in the plastic around the connector. When the device is connected to a host, the transmitter is enumerated along with the user's expected interfaces. Software running on the host can then exfiltrate data or receive commands via the RF interface. Identifying and mitigating these additional interfaces has traditionally been difficult as an adversary can simply reprogram any USB descriptive data (e.g., VID and PID) to evade device whitelisting or blacklisting rules in the operating system. Furthermore, mitigation is complicated by the legitimate use of composite devices such as audio headsets with both input and output.

Devices do not have to be equipped with new hardware components to be evil. The lack of authentication for firmware in USB devices allows attackers to overwrite the firmware with malicious code [15]. Devices infected with BadUSB [83], for example, can present malicious interfaces as simple as a HID interface or as complex as a network adapter. iSeeYou [24] modifies a webcam's firmware to disable the indicator light. Psychson [27] modifies the firmware of a USB storage device by adding a keyboard functionality, which can run the malicious script automatically. These attacks are invisible to the user and the resulting modified device can be moved between hosts, leaving a number of host machines exploited.

2) Protocol Corruption: The host's USB software stack generally expects devices to conform to the USB standard. Fuzzing techniques using FaceDancer [49], [101], [64] and debuggers [16] have led to the discovery of a number of kernel-mode arbitrary code execution vulnerabilities, e.g., in the Windows USB drivers [2], [3], [4], FreeBSD [23], Linux kernel USB subsystem [11], [97], [20], and all other operating systems [38]. In 2017, syscall fuzzer syzkaller also found more than 40 bugs in different Linux kernel USB drivers [50]. In some cases, exploitation of these vulnerabilities can occur during the host's device enumeration, making the physical connection of the device the only barrier to compromise. Man-in-the-middle devices such as embedded systems running USBProxy [41] can manipulate legitimate protocol traffic from devices to inject malicious content.

D. Abuse of Physical Layer

Physical layer attacks consist of attacks against confidentiality and integrity in the communication across the USB bus.

1) Signal Eavesdropping: In signal eavesdropping attacks, sensitive data is recovered through physical observation of messages moving between the host and peripheral. Keyloggers are miniature, inconspicuous shim devices that are placed between the host port and peripheral to record keystroke packets, e.g., KeyGrabber [68]. Shah et al. present JitterBug [102], a *single trip* keylogging attack that exfiltrates keystrokes from the target over a timing-based network side channel. Neugschwandtner et al. demonstrate that, prior to USB 3.0, a malicious peripheral can eavesdrop on the downstream traffic of all connected devices [81]. USB snooping [108] attacks leverage current leakage on the power line of the USB bus to infer the USB data traffic. There have also been in-the-wild appearances of malicious USB peripherals and cables that use network connectivity to eavesdrop and exfiltrate sensitive messages, such as CottonMouth [6], [7] and GPS locator [77], [12]. Of particular concern is the fact that many hosts contain internal USB hubs which are often reprogrammable [82], allowing for a persistent bus eavesdropping compromise via firmware rewriting regardless of BIOS or UEFI integrity defenses.

A variety of fingerprinting attacks have also been demonstrated in which low layer messages are shown to leak significant information about host characteristics. Wang and Stavrou demonstrate that USB Request Blocks leak information about the host operating system [125], which can be used by a malicious smart phone to compose a targeted malware payload. Davis observes that variations in the implementation of USB enumeration protocol can be used to identify the operating system, e.g., Windows 8 is the only common operating system to issue 3 `GetConfiguration` descriptor requests [39]. A more resilient approach to host fingerprinting relies on timing side channels (e.g., inter-packet gaps) to infer host machine characteristics. Letaw et al. [72] employ a USB protocol analyzer [43] to extract timing features of bus states and use machine learning classification to infer the operating system of the host.Bates et al. present a timing-based fingerprinting scheme that can be launched from a commodity smart phone. They show that specific operating system versions and model numbers can be inferred with upwards of 90% accuracy, that inter-packet gaps can be used by devices to detect the presence of virtualized environments [17]. While timing-based fingerprinting significantly raises the bar for evasion, it seems likely that resource-rich hosts could modify their timing characteristics to evade detection, although this has not been demonstrated in the literature.

2) Signal Injection: Analog signals are used to convey sensitive data, leaking information to the outside of the machine, where an adversary is able to receive the signal, decode it, and recover the sensitive data. For USB, the most common covert channels are based on electromagnetic emission [106] and power analysis. Unlike USB bus eavesdropping mentioned above, USBee [51] does not require any specific devices or cables to leak the data from the host machine. Instead, it uses connected USB devices as an RF transmitter to emit electromagnetic emissions that encode sensitive data. Where there is no "victim" RF transmitter available on the laptop, the adversary can touch the exposed metal part of the machine with a plain wire, the signal of which can be used to extract secret keys [48]. Similarly, Oswald and etc. [87] show how to extract AES 128-bit keys from Yubikey 2 based on power consumption and electromagnetic emanation.

The ability to inject analog power has also been used to cause physical damage to the host machine. USB Killer (and USB Kill 2.0) [120] embeds a number of capacitors on the two sides of the PCB board of the USB key. Once connected with the host machine, USB Killer draws the power from the host USB bus, charging the capacitors. Once fully charged, a negative 200VDC is discharged over the USB data lines of the host machine. This charge/discharge cycle keeps going until the USB Killer is removed or the host machine is damaged. In newer releases of USB Power Delivery and Type-C connector standards, device are able to draw and transmit so much power, e.g., up to 100W, that they can irreparably damage the host. The use of poor quality USB Type-C cables have already led to benign circumstances that resemble this attack [19], where a cable has damaged a Pixel book and two USB PD analyzers.

USB Attack Vector Summary

Based on this survey, we identify several *offensive primitives* that are being leveraged in USB-based attacks. Table I provides a mapping of notable attacks surveyed above to the layer and primitive to which they correspond. From the above taxonomy of attacks, the following findings can be drawn.

F1. Trust by Default: Across all communications layers, a common characteristic of attacks is that they abuse the trust-by-default assumption that pervades the USB ecosystem. This trust model is inextricably linked to the "Plug & Play" philosophy that led to USB's ubiquity, making popular the notion that peripherals should work instantly upon connection without any additional configuration. At the physical layer, USB 1.x and 2.x broadcast downstream messages assuming they would only be read by the recipient. At the transport layer, USB protocols assume that kernel drivers will only be requested for legitimate purposes. Within the application layer, host machines blindly trust the integrity of the contents of portable media and devices assume that all transactions emanate from a trustworthy agent. Violations at the human layer are the result of misplaced trust in the intentions of devices and other humans.

Unfortunately, trust-by-default is not strictly a legacy problem. As recently as late 2014, the USB-IF stated that "consumers should always ensure their devices are from a trusted source and that only trusted source interact with their devices" [117]. The assertion that the consumer is responsible for the integrity of the USB interaction is problematic; consumers have no means of establishing the identity or provenance of a

Layer	Offensive Primitive	Attack
Human Layer	Outsider Threats	Social Engineering USB [107], U.S. Government [89], USB Attack Vector [61], Users Really Do [115]
	Insider Threats	Ponemon Study [75], Australian Defense Loss [37], Manning Infiltration [69], Snowden Documents [10]
Application Layer	Code Injection	Brain [58], Stuxnet [30], Conficker [104], Flame [127], User-mode rootkit [109]
	Data Extraction	Webcam Extraction [85], [1], Audio Extraction [110], USBee [51]
Transport Layer	Protocol Masquerading	Rubber Ducky [52], USBdriveby [66], TURNIPSCHOOL [5], USB Bypassing Tool [15], BadUSB [83], iSeeYou [24]
	Protocol Corruption	FaceDancer [49], Syzkaller [50]
Physical Layer	Signal Eavesdropping	Smart Phone USB Connectivity [125], USB Stack Interaction Intelligence [39], Host Identification by USB Fingerprinting [72], [17] USB Eavesdropping [81], USB Snoop [108], CottonMouth [6], [7], USB GPS locator [77], [12], BadUSB Hubs [82]
	Signal Injection	USBKiller [120], Cable Quality [19], Power and EM Covert Channels [106], [87], USBee [51], Plain Wire Signal Attacks [48]

Table I: Notable real-world attacks on the peripheral ecosystem, grouped by the layer at which they operate and the offensive primitive of which they are an instance.

device, making it impossible to determine if it originates from a trusted chain of custody.

F2. Attacks Transcend Layers: As demonstrated through the survey of attacks, attacks that exploit hosts or exfiltrate data from them appear to demonstrate correct operation to the layer they are communicating with. For example, attacks such as USBee allow the passing of messages that look for all intents and purposes like legitimate traffic, or at least traffic that is allowable within the USB standard, while the actual exfiltration is a physical layer activity based on RF or GSM emanations. Similarly, attacks such as BadUSB and TURNIPSCHOOL are not subverting the USB protocol itself, but exploiting its inherent openness to augment functionality that users would not think to look for. The consequence of this is that solutions that simply consider one particular segment of USB activity without adopting a more holistic approach to the entire USB stack, both in terms of its hardware and software, will be necessarily incomplete and potentially susceptible to these cross-layer attacks.

IV. Securing USB

Defenses are organized based on the layer at which attacks target, not on the layer of the system that they modify to provide the defense. For example, on-device encryption is a low-layer solution to defending against a human-layer problem (data loss). In some cases, individual systems feature defensive mechanisms for multiple operational layers; we discuss these in multiple subsections below. As mentioned earlier, the derived defensive primitives describe both the mechanism employed as well as the security properties guaranteed.

A. Defense of Human Layer

For defenses targeting the human layer, we divide solutions into those that impact the capabilities of human stakeholders, mechanisms that operate on device (such as encryption and authentication), and auditing mechanisms either on the host or the device itself.

1) Security Training: Perhaps the most difficult challenge to USB-based attacks is mitigating attempts to "hack the human." A necessary first step to prevent peripheral attacks in security-sensitive organizations is extensive and frequent security training. In 2012, NIST set out standards for using portable devices including USB [124], and these standards are also making their way into many organizations' security education programs. Increasingly, employees are made aware of the dangers of social engineering [8]. After security training sessions, lessons are commonly reinforced through mounting informational security posters around the workplace that warn of social engineering tactics, e.g., [28]. Still, in a survey done by CompTIA, 45% of employees report having received no corporate security training whatsoever [33]. To make matters worse, empirical evaluation has shown that security training is not a panacea for security illiteracy [70], [103], and anecdotal evidence indicates that skilled social engineers are capable of assuaging the reservations of their targets even after security training [42].

2) On-Device Data Encryption: Encrypted USB devices (e.g., IronKey [60] and Kanguru [67]) provide data confidentiality through on-device encryption and user authentication, and employ tamper-resistant hardware to prevent physical extraction of data or keys. By encrypting data stored on removable media, these devices prevent the loss of data through physical theft of the device. While relatively costly in comparison to standard USB storage devices, these have seen considerable industry adoption, at the price of complicated device enrolling and key management processes. Even when encrypted, however, on-device encryption can not prevent data loss due to insider attacks. Diwan et al. [40] achieve functionally equivalent properties to on-device encryption by instrumenting the Windows USB subsystem to perform on-the-fly encryption of outbound I/O request packets. This approach requires invasive modifications to the host operating system and lacks the portability of secure flash drives, but can prevent data exfiltration via USB as hosts outside of the organizational boundary will be unable to read the device.

3) On-Device Host Authentication: In response to emerging peripheral attack vectors, recent proposals have sought to bind device functionality to particular machines rather than specific users. The Kells system [25] extends USB enumeration to support host identification via trusted hardware. Kells assumes the presence of a TPM on the host as well as a custom TPM daemon, and introduces a custom smart USB device. Following the end of standard enumeration, a full TPM attestation is performed over the USB interface using Acceptance Device Specific Command (ADSC). If the device successfully verifies the host TPM's quote result then all partitions are mounted, otherwise only a public partition is mounted. This approach to host identification is also used in the ProvUSB system [113]. Host-identifying smart devices can therefore prevent data loss due to both device theft and insider attacks, as in either case the attacker will be unable to access the data partition on an unauthorized host. However, these systems require a number of extensions to the standard connectivity model including modifications to both the host and the device, trusted hardware, and a security policy for whitelisting host access.

4) Host- or Device-Based Auditing: In the absence of a foolproof method for securing the human layer, a viable alternative is auditing peripheral usage. Auditing provides system administrators an opportunity to reason about how peripherals are being used within the organization, e.g., allowing them to monitor the flow of data via flash storage drives in much the same way that network monitoring software grants the ability to track data entering and exiting the organization over the Internet. Techniques have been demonstrated to recover evidence of portable media usage from the host in spite of the *anti-forensic* properties of USB flash drives [22], [74], although these approaches are susceptible to false evidence presented by malicious peripherals [95]. Extending the host operating system with provenance-based auditing capabilities [18], [65] has been shown to be useful when attempting to identify the root cause of data exfiltration attacks. By recording when data is written to storage device, data provenance can narrow the list of suspects if sensitive data is discovered in a public forum. The ProvUSB system permits fine-grained audit data to be collected on board [113].

B. Defense of Application Layer

Defenses targeting attacks at the application layer focus primarily on the host, and include modifications to the OS and its drivers.

1) System Hardening: Host systems can also be hardened through enabling safer default behaviors. Pham et al. [90] inspect Windows OS families and re-configure the system to disable auto-run-like functionality and block the execution of unsigned executables or drivers carried on portable media. Antivirus software can also be used to prevent application layer attacks over USB storage. Composite anti-virus systems such as Metascan [86] and OLEA [84] offer not only standard malware scans for host machines, but also sell scanning kiosks in which sacrificial VMs are used to ensure containment of any malware. These kiosks are commonly deployed near the entrances of security-sensitive organizations to prevent infected peripherals from entering the facility. The Windows Embedded platform [79], TMSUI [126] and USB Unix Smart Blocker [40] attempt to mediate USB connectivity for Windows CE, Industrial Control and GNU/Linux Systems respectively, but all base their device recognition mechanism on unreliable information reported by the device during enumeration. USBFILTER [112] instruments the upper layers of the USB stack, modifying device drivers in order to identify the processes interactive with the device. USBFILTER can thereby *pin* devices to specific process ID's, creating a novel defense against application-layer attacks in which malware eavesdrops on USB device traffic to obtain sensitive information (e.g., keystrokes, webcam images).

2) Driver-Based Access Controls: Treating USB drivers as "capabilities", GoodUSB [112] system attempts to constrain malicious peripherals through incorporating elements of user-driven access control [96] for driver loading. Prior to the completion of enumeration, GoodUSB reports the device's claimed identity to the user via a pop-up notification. Based on the user's expectations of device functionality, GoodUSB then permits all or some of the requested driver's to be loaded on behalf of the device; for example, when the user expects a peripheral to be a flash drive, the peripheral will not be able to request the Human Interface Device driver during enumeration. Because authorization is based on requested behaviors instead of reported identity, GoodUSB cannot be circumvented by a malicious device, thus defeating BadUSB attacks. However, it cannot prevent peripherals from making malicious use of their natural drivers (e.g., a malicious keyboard injects keystrokes).

3) Device-Emulating Honeypots: Various strains of advanced malware are now known to attempt to propagate to and from hosts and storage devices. Frequently, the malware will wait for a peripheral connection and then attempt to propagate to the other end of the connection shortly thereafter. As a result, honeypots have been demonstrated to be an effective means of detecting the presence of an infection. Host-emulating honeypots such as Ghost can detect the propagation of malicious USB storage payloads [91], which it accomplishes through emulating a storage device that periodically connects to potentially infected machines. If the host initiates any file I/O with the emulated device, this is likely evidence of malicious activity. as under benign circumstances the host will not interact with the dummy device after SCSI scanning.

C. Defense of Transport Layer

Defenses against attacks in the transport layer are broken down by firmware verification, USB stack fuzzing, USB packet firewall, and host-emulating honeypots.

1) Firmware Verification: Secure USB devices such as IronKey purport to prevent BadUSB attacks [59] by using signed firmware, provided that the device manufacturer is trusted and the signing key is kept safe. While signed firmware is a sound practice, the introduction of a trusted third party expands the attack surface of the system. When the device firmware is accessible, e.g., via Device Firmware Update (DFU), FirmUSB [54] applies symbolic execution to find hidden and malicious functionalities inside the firmware. Unfortunately, FirmUSB only supports 8051 architecture, leaving ARM Cortex-M and AVR based firmware uncovered. Also, the firmware is often not available, even in binary format. Another way to verify the firmware is via attestation, where the host verifies the correctness of device firmware by establishing tight timing bounds on its response to a series of challenges. VIPER [73] presents a software-based timed challenge-response protocol for verifying peripheral firmware over the system bus that precludes the possibility of proxy attacks by leveraging the asymmetry of the latencies from CPU-to-peripheral and from peripheral-to-proxy. In spite of the known difficulty of performing software-based attestation on embedded devices [26], this approach is practical for

Layer	Defensive Primitive	Defense
Human Layer	Security Education	NIST Standards [124], Education Materials [8]
	On-Device Data Encryption	IronKey [60], Kanguru [67]
	On-Device Host Authentication	Kells [25], ProvUSB [113]
	Host- or Device-Based Auditing	System Provenance [18], Transient Provenance [65], ProvUSB [113]
Application Layer	System Hardening	Disabling Autorun [90], Metascan [86], OLEA [84], WindowsCE [79], TMSUI [126], Smart Blocker [40], USBFILTER [114]
	Device-Emulating Honeypots	Ghost [91]
	Driver-Based Access Controls	GoodUSB [112]
Transport Layer	Firmware Verification	IronKey [59], FirmUSB [54], VIPER [73]
	USB Stack Fuzzing	USB Fuzzing [80], [121], Hardware-based Fuzzing [62], vUSBf [100], Syzkaller [50], POTUS [88]
	USB Packet Firewall	USBFILTER [114], USBFirewall [63]
	Host-Emulating Honeypots	GoodUSB [112], Cinch [13]
Physical Layer	Anti Fingerprinting	USB Host Fingerprinting [17]
	Secure Channel	Cinch [13], UscramBle [99]

Table II: Proposed defenses for the peripheral ecosystem, grouped by the layer at which they defend and the primitive of which they are an instance. Note that many solutions employ multiple defensive primitives.

pluggable peripherals due to the physical connection between the challenger and responder, as long as the timing profile for each device exits and is trusted.

2) USB Stack Fuzzing: USB fuzzing has long been incorporated into security consultants' threat assessments [80], [121]. Jodiet et al. present a mutation-based USB fuzzing approach that is conducted on hardware using a PCI evaluation board and the Linux USB Gadget API [62]. Schumilo et al. present a QEMU-based, parallizable virtual USB fuzzer (vUSBf) that makes use of USB redirection to inject arbitrary noise into different `GetDescriptor` requests [100] Leveraging KCOV feature within the Linux kernel and QEMU, Syzkaller [50] is a coverage-guided syscall fuzzer, and has shown its usefulness to find bugs in the USB subsystem. POTUS [88] combines fault injection, fuzzing, and symbolic execution to detect bugs in USB kernel drivers. While fuzzing can improve the code quality and raise the bar for attackers, it cannot defend against attacks abusing the USB protocol itself, such as BadUSB attacks.

3) USB Packet Firewall: As network firewalls are a powerful primitive for minimizing the potential actions of would-be attackers on the Internet, firewall-driven protocol access controls for USB peripherals intuitively provide similar protections. Tian et al. present USBFILTER/usbtables [114], a netfilter/iptables-like stack for filtering USB traffic. Where iptables enforces rules by pattern matching over IP addresses and port numbers, usbtables can pattern matches USB buses and ports, among other fields; these correspond to physical locations on the host machine that cannot be spoofed by a malicious peripheral. USBFILTER can then apply rules that constrain permissible protocol activities in much the same way as GoodUSB. USBFirewall [63] is another USB packet firewall implementation upon FreeBSD. Unlike USBFILTER, USBFirewall focuses on protecting the host USB stack by detecting malformed USB packets, e.g., generated by FaceDancer. We believe the combination of USBFILTER and USBFirewall should provide a stronger protection for USB host security.

4) Host-Emulating Honeypots: In contrast with device honeypots, which can only detect malware propagating from a host to portable storage, emulating the host machine allows for the detecting of malicious peripheral activity at both the application and transport layers. To examine a suspicious device, GoodUSB [112] redirects it to a QEMU-KVM virtual machine using USB pass-through. The VM completes the USB enumeration and then monitors the device for evidence of malicious activity. The Cinch system [13] also leverages virtualization to decrease the host's attack surface – the host operating system is hoisted into a VM to isolate it from the USB host controller, and then all USB traffic via is tunneled via IOMMU through a sacrificial gateway VM. Within the sacrificial VM, a variety of the application and transport layer defenses techniques can be deployed including signature-based antivirus, protocol compliance, and user-drive access control. While host honeypots are able to detect both application and transport layer attacks, the VM dependency and device operation interruption make them impractical for normal users.

D. Defense of Physical Layer

Defenses against physical layer attacks are organized by anti fingerprinting as well as implementing confidential communication over the USB bus.

1) Anti Fingerprinting: Solutions for physical layer attacks have received only limited consideration within the literature. The most straightforward means of mitigating inferences from fingerprinting attacks is to further randomize the USB stack behavior in hardware and software. A technique for defeating message-based fingerprinting is demonstrated in [17] by introducing additional `GetDescriptor` requests to confuse the attacker; generalized, this result indicates that uniform appearance and ordering of control transfers during USB enumeration will make it more difficult to distinguish between operating system families. Timing randomization is also possible to defeat timing-based fingerprinting. As a host/device can arbitrarily speed up or slow down USB transfers, it could confuse would-be attackers by varying its timing characteristics. However, as the USB spec imposes requirements regarding message ordering and timing, extreme behavior randomization may break the normal operation of the device.

2) Secure Channel: To defend against USB bus eavesdropping, Cinch [13] considers adapting encryption and authentication schemes to the physical peripheral connections. A Cinch gateway is used as an encryption and decryption proxy on the host side, and a small crypto adapter (similar to a keylogger) to act as the peripheral's encryption and decryption proxy. As a result, a malicious USB bus or other

USB devices would only have access to encrypted traffic and could not produce authenticated messages. Similarly, the UScramBle [99] system defends against eavesdropping of downstream traffic by instructing the host to negotiating an encryption key with the host during USB enumeration.

SECURING USB SUMMARY

Based on this survey, we identify several *defensive primitives* that are being leveraged in USB security solutions. Table II provides a mapping of notable defenses to the layer and primitive to which they correspond. Further, from the above taxonomy of defenses, the following findings can be drawn.

F3. Trust Anchors represent a Design Tradeoff: The intrinsic flaw enabling all offensive primitives uncovered by our analysis was the misguided *trust-by-default* property underlying the USB ecosystem – both the host and the device are assumed to be benign and expose all functionality to one another after enumeration. It is therefore not surprising that the majority of viable defensive primitives require the introduction of a trust anchor in order to enable their security properties. Smart device prototypes such as Kells [25] and ProvUSB [113] propose the use of host-side trusted hardware for authentication, while commercial solutions like IronKey [60] verify user-presented credentials. One notable consequence of the trust placement design tradeoff is that the placement of the trust anchor (host vs. device) informs the directionality of the defense. Smart devices seeking to defend themselves from malicious actors on the host leverage host-side trusted hardware prior to granting access. Host machines, in turn, anchor trust in the intrinsic physical properties of USB device firmware in order to defend against malicious peripherals. Based on this observation, it is clear that *a complete solution to USB security will likely require trust anchors on both the host and device sides*.

F4. Single-Layer Solutions Are Not Effective: An emerging trend [13], [114], [63] in the recent literature is that threats in the USB peripheral space can be understood through the lens of network security – by presenting peripherals to the host as untrusted network endpoints, the host will be able to defend itself from attack. The primary examples of this primitive are USBFILTER [114] and USBFirewall [63]. As shown in Table III, this primitive is proven to be the most powerful solution, covering attacks across different layers. Similar to the firewall primitive, host-emulating honeypots, such as GoodUSB [112] and Cinch [13] also expand their defense into different layers. The power of these solutions is rooted in the fact that they are a composition of protection mechanisms within different layers that provide different operational semantics. Based on our analysis of the USB defense space, we conclude that *a complete solution must be able to centralize context from all operational layers prior to issuing security decisions*.

F5. Defenses for Signal Injection Are Still Missing: As shown in Table III, there is still no defense primitive available to defend against signal injection attacks based on our analysis. These attacks usually leverage the intrinsic nature of hardware as side channels to emit analog signal, such as USBee [51], or require hardware changes for power attacks, such as USBKiller [120]. It is natural to see why software-based solutions could not mitigate these attacks. While USB hardware design improvement is definitely the right direction in the long run, we need a way to establish trust with USB devices before fully enabling them in the short term, such as disabling bad-quality charging cables [19].

V. IS USB TYPE-C THE ANSWER?

As we have explored, the research community has proposed many different solutions for addressing weaknesses in USB security, however none have reached widespread, commercial adoption. In this section, we evaluate the industry's proposed solution, USB Type-C Authentication [116]. Type-C Authentication (TCA) is the first attempt by the USB 3.0 Promoter Group and USB-IF to address issues related to security. However, the security properties of TCA are not yet widely understood by the security community[1]. We begin with a description of the features and assumptions of TCA. Then, using the Type-C Authentication revision 1.0 specification (released on Feb 2, 2017), we formally model and verify the protocol using ProVerif [21], demonstrate multiple attacks, and discuss other issues within the spec. We finally evaluate TCA using findings we have learned through the systematization, and show that TCA is on the right direction to solve USB security in general, but the design flaws and the ignorance of modern USB attacks render its efforts in vain.

A. TCA Description

1) USB Certificate Authorities: The TCA protocol is built upon a certificate authority (CA) hierarchy, mimicking the current CA model used with SSL/TLS connections on the Internet. However, in this case, the USB-IF owns the default self-signed root certificate, and permits other organizations to use their own root certificates. The specification makes no requirements on third-party roots (e.g., organizational vetting or issuance processes). USB device manufacturers provide intermediate certificates signed by the USB-IF, and devices are issued their own certificates by the manufacturers. Certificates follow the X.509v3 format with DER encoding. The final USB product is capable of storing at most 8 certificate chains and associated private keys, each with separate roots.

2) Authentication Protocol: In this protocol, the initiator is the USB host controller and the responder is the USB device. The protocol defines three operations the initiator can perform, shown in Figure 4:

Digest Query: In this operation, the host controller issues a `GetDigest` request to the device. The device responds with digests for all its certificate chains. According to the specification, the intent of this operation is to accelerate the certificate verification process in cases where the certificate chain has already been cached and verified.

[1] At the time of writing, the only commercial products supporting TCA are software from Siliconch [105] and a USB PD controller from Renesas [94].

	Outsider Threats	Insider Threats	Code Injection	Data Extraction	Protocol Masquerading	Protocol Corruption	Signal Injection	Signal Eavesdropping
	Human Layer		Application Layer		Transport Layer		Physical Layer	
Security Training	◐	◐	–	–	–	–	–	–
On-Device Data Encryption	◐	–	–	–	–	–	–	–
On-Device Host Authentication	◐	□	–	–	–	–	–	–
Host- or Device-Based Auditing	–	□	□	□	–	–	–	–
System Hardening	–	–	◐	◐	–	–	–	–
Driver-Layer Access Controls	–	–	◐	◐	–	–	–	–
Device-Emulating Honeypots	–	–	–	□	–	–	–	–
Firmware Verification	–	–	–	–	◐	◐	–	–
USB Stack Fuzzing	–	–	–	–	–	◐	–	–
USB Packet Firewall	–	–	◐	◐	●	●	–	–
Host-Emulating Honeypots	–	–	■	–	■	■	–	–
Anti Fingerprinting	–	–	–	–	–	–	–	◐
Secure Channel	–	–	–	–	–	–	–	●
Type-C Authentication	–	□	□	–	–	□	◐	–

Table III: Comparative evaluation of defensive primitives for securing the USB stack. Columns represent offensive primitives as organized by the communications layer. Defensive primitives are marked with ● if they provide a complete defense, ◐ if they provide a provide a partial defense, ■ if they can reliably *detect* that an attack has taken place, and □ if they provide detection under limited conditions.

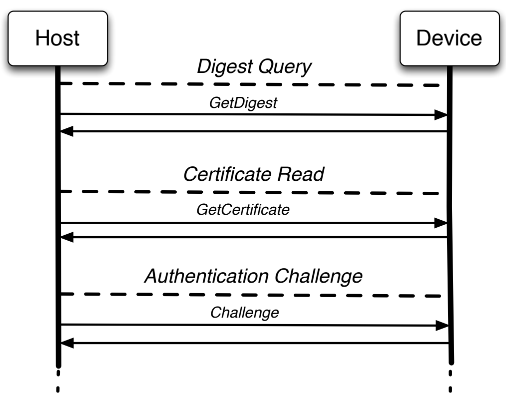

Figure 4: The USB Type-C Authentication Protocol.

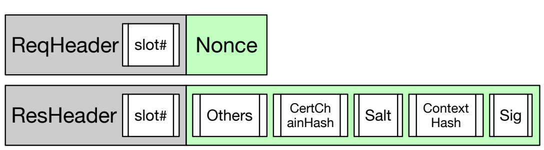

Figure 5: The USB Type-C Authentication challenge (request) and response messages with payloads.

Certificate Read: This operation allows the host to retrieve a specific certificate chain using the `GetCertificate` request.

Challenge: As shown in Fig 5, this operation defines a challenge-response protocol where the host initiates by sending a `Challenge` request. The request contains a slot identifier in the request header and a 32-byte nonce. The response echoes the same slot identifier in the response header and contains a 32-byte SHA256 hash of the chosen certificate chain, a 32-byte salt, a 32-byte SHA256 hash of all USB descriptors for USB devices and all zeros for PD devices, and a 64-byte ECDSA digital signature on the challenge message and the response message using the corresponding private key of the device.

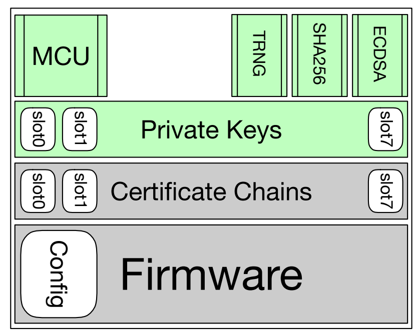

Figure 6: USB device internal architecture with secure storage and hardware to support Type-C Authentication.

3) Secure Key Storage and Processing: To protect certificates' private keys, a non-volatile secure enclave is needed, shown in Figure 6. As discussed above, this storage is partitioned into 8 slots supporting 8 private keys. Similarly, the certificate chain region also has 8 slots, containing the corresponding certificate chain if there is a private key in the associated slot. The TCA specification does not specify whether certificate chains should also be secured.

To support the authentication protocol, a hardware cryptographic engine supporting ECDSA is also required. Presumably, this should be the only component which can access the secure storage. Other hardware components, besides the basic MCU, may be needed for both security and performance reasons, including TRNG and SHA256.

4) Security Policy: Following device authentication, the TCA specification suggests the introduction of a policy mechanism for peripheral management. The specification explains that "Policy defines the behavior of Products. It defines the capabilities a Product advertises, its Authentication requirements, and resource availability with respect to unauthenticated Products" (Page 14, Section 1.4) and "USB Type-C Authentication allows an organization to set and enforce a Policy with regard to acceptable Products." (Page 11, Section 1). Unfortunately, beyond this description a concrete definition for policy is not provided; all implementation details are left to the OEM.

B. Formal Verification

To discover possible vulnerabilities in the design, in this section we formally verify the TCA protocol using ProVerif [21]. ProVerif uses the concept of *channels* to model an untrusted communication environment (e.g., the Internet) where adversaries may attack the protocol. However, because the USB communication channel does not provide confidentiality by default and is trusted in most cases[2], we instead model the device firmware as our channel. This accurately models attacks such as BadUSB [83], where the attacker is either a malicious USB device or a non-root hub trying to spoof the authentication protocol. In ProVerif, we define this firmware channel as `free fw:channel.`

We also need to define the security properties we wish to prove. For example, since the private keys inside USB devices should never be leaked, we seek to understand if attackers can learn the key from eavesdropping or participating in the protocol. The Type-C authentication spec also says clear that "it permits assurance that a Product is

1) Of a particular type from a particular manufacturer with particular characteristics
2) Owned and controlled by a particular organization".

The means the authentication protocol should guarantee both the original configuration and the true identity of the device. The original configuration should be the one designed by the vendor for this product (e.g., a webcam). The *true identity* combines the usage of certificate chains (tying to a particular organization) and private keys baked into the device to provide the ability to cryptographically verify the original configuration We abstract these security goals in ProVerif:

```
free slot_key:pri_key [private].
free slot_cert_chain:cert_chain.
free orig_conf:usbpd_config.
query attacker(slot_key).
query d:usbpd; event(goodAuth(d, true)) ==>
    event(useConfig(d, orig_conf)).
query d:usbpd; event(goodAuth(d, true)) ==>
    (event(useCert(d, slot_cert_chain)) &&
    event(usePrivkey(d, slot_key))).
```

To simplify the abstraction, we model one private key and the corresponding certificate chain rather than implementing all 8 slots. We also make the following assumptions:

- We ignore the verification process for a certificate chain, which is critical to the security of the entire protocol but out of the scope of the protocol.
- We assume the verification process to be successful by default.

[2]We do not consider side-channel or hardware attacks against the USB bus.

Our modeling is based on the communication between the USB host and the USB device. PD products share the same procedure via different signaling mappings. To mimic the caching behavior involved in the protocol, we use a "table" in the host side, supporting reading and writing a certificate chain: `table cert_chain_cache(cert_chain, digest).`

C. Results

Unsurprisingly, attackers cannot obtain the private key inside the USB device by protocol messages alone since none of these messages are designed to transmit the key. However, this protocol fails to meet its goals; neither the original configuration nor the true identity of the device could be guaranteed even if the authentication protocol succeeds, due to the usage of certificate chain cache inside the USB host:

```
get cert_chain_cache(chain, =dig) in known_device(config)
    else new_device(config).
```

Since the certificate chains are not secret, a malicious device can compute the digest of the expected chain. This digest can be sent as a response to the `GetDigest` request and impersonate the legitimate device. Unless the configuration of the legit device is saved and checked by the host, a malicious device can claim any functionalities it wants. Thus, **the certificate chain cache is vulnerable to spoofing attacks**.

We then remove the certificate chain cache from the host, forcing every device to go through a complete certificate request. Again, the private key is secure. Unfortunately, the authentication can still be spoofed as shown in this attacking trace:

```
attacker(sign((non_1883,hash(chain_1877),sal_d_1881,
    config_d_1879),prik_1876)).
```

To exploit this vulnerability, the attacker hardcodes a certificate chain and a private key in the firmware rather than using the ones in the slot and modifies the original configuration (e.g., by adding a malicious HID functionality). This means that **without firmware verification to prevent BadUSB attacks, these also allow circumventing the TCA protocol**, *rendering it useless for its stated goals*.

To demonstrate how firmware verification corrects this issue, we then assume firmware is trusted (e.g., signed by the vendor and verified by the MCU before flashing). We model this in ProVerif by marking the firmware channel as private: `free fw:channel [private]` We assume that valid, legitimate firmware will use the certificate chains and private keys inside the slots during authentication and that the original configuration of the device does not contain malicious functionality.

Using this model, ProVerif confirms that successful authentication guarantees both the original configuration and the true identity of the device:

```
RESULT event(goodAuth(d,true)) ==>
    (event(useCert(d,slot_cert_chain[])) &&
    event(usePrivkey(d,slot_key[]))) is true.
RESULT event(goodAuth(d_2076,true)) ==>
    event(useConfig(d_2076,orig_conf[])) is true.
RESULT not attacker(slot_key[]) is true.
```

These results show that correct authentication using the TCA protocol is possible only when the firmware is verified.

D. Other Issues

While out formal verification of the authentication protocol uncovered major flaws, our manual analysis of the TCA specification uncovered other serious and systemic design flaws. These flaws reflect both a lack of understanding of secure protocol design and a lack of awareness to the present state of threats to peripheral devices. Responsibility for solving the most difficult security challenges raised by Type-C, such as a USB Certificate Authority system or a rich language for expressing security policies, is delegated wholesale to the OEMs. As a result, we are left to conclude that Type-C is based on an intrinsically broken design. Below, we catalog these issues:

1) **No Binding for Identification with Functionality:** In addition to the VID, PID, and serial number of the device, a device's leaf certificate also carries Additional Certificate Data (ACD). ACD contains physical characteristics of PD products (e.g., peak current and voltage regulation) but no functionality (interface) information for other USB products. One explanation is that the protocol was designed to address bad-quality Type-C cables that were damaging host machines [19] but was later extended to support other USB products. For PD, the specification clearly states that it does not consider alternative modes. As a result, a successful authentication does not specify the device's original configuration (e.g., storage device, keyboard, normal charging cable).
2) **Volatile Context Hash:** As shown in Fig 5, the challenge response contains the context hash, which is all zeros for PD products but a SHA256 hash of all descriptors for USB products. This seems intended to solve the functionality binding issue for USB products mentioned above but is broken when the firmware is not trusted. However, the firmware can provide its own set of USB descriptors, and feed them into the hardware ECDSA signing module to generate the challenge response, as shown in Fig 6. As a result, BadUSB attacks are still possible.
3) **Unidirectional Authentication:** For PD products, either a PD sink or a PD source can initiate an authentication challenge. This means the authentication between PD devices is mutual. However, the TCA specification only allows for USB host controllers to initiate an authentication challenge for USB devices. This is unfortunate, as our survey of defensive solutions demonstrates that host authentication is an essential feature for device self-protection. As a result, the TCA specification does not provide a way for smart devices such as cellphones to make informed trust decisions.
4) **Nebulous Policy Component:** Following device authentication, the TCA specification calls for the creation of a security policy in order to handle different connected products, but does not adequately describe what a policy is or how to create one. The specification does not define the security policy language, encoding, installation method, or how it interacts with the USB host controller. Policies are only described anecdotally, indicating a lack of forethought as to how TCA policy can be used to appreciably enhance security.
5) **Impractical Key Protection Requirement:** The private keys in the slots are the most important property a product needs to protect besides the firmware. Although the specification does not detail how to secure private keys, it does list more than 10 attacks a product needs to defend against from leaking keys, including side-channel attacks, power analysis, micro-probing, etc. It is unlikely that a $10 USB product [94] could stop advanced invasive attacks, e.g., using Focused Ion Beam (Appendix C, TCA Spec), which makes certificate revocation critical when a private key is leaked…
6) **No Revocation:** The specification states that the validity time of a product certificate is ignored, suggesting that once the certificate is loaded onto the device, there is no way to revoke it. The usage of certificate chain cache to accelerate the authentication process is also based on the fact that all certificates along a chain stay legit forever once the chain is verified.
7) **No Support for Legacy Products:** With the help of converters, Type-C can be fully compatible with legacy USB devices, and leaves it to the end user to set a security policy that blacklists devices that cannot participate in the authentication protocol. As breaking backwards compatibility is in direct conflict with the USB's core design principle of *universality*, very few organizations will elect to set such a policy. As a result, TCA is likely to be trivially bypassable by applying a converter to a Type-C device.

We map TCA as a new defense primitive against all attack primitives in Table III, which shows the limitation of TCA as a complete USB security solution. Not surprisingly, TCA works best for signal injection attacks since it was designed to solve bad-quality charging cables. All other limited defense effects are the results of trusting the identity and the firmware once the device passes the authentication protocols, and assuming some security policies deployed on the host machines using the identity of the device.

We then evaluate TCA using all the findings based on our systematization, as shown in Table IV. One one hand, TCA is aware of some urgent issues in USB security, takes initial steps, and tries to fix them. For instance, TCA introduces the CA model and certificates providing a way to build trust for USB products, and embeds private keys into USB products providing trust anchors. On the other hand, as we show in the TCA weakness column, the design flaws and limitations makes TCA a vulnerable and incomplete solution for USB security.

VI. Future Directions

Through systematization, we have demonstrated that a complete solution requires a system that composes multiple defensive primitives, across different communication layers.

Finding	TCA Strength	TCA Weakness
F1. Trust by Default	CA model & Certificates	1. Certificate chain cache; 2. Firmware implementation; 3. No support for legacy devices
F2. Attacks Transcend Layers	N/A	1. Depending on nebulous "Policy"
F3. Trust Anchors As Design Tradeoff	Private keys	1. Unidirectional authentication; 2. Key protection requirement; 3. No revocation
F4. Single-Layer Solutions Are Not Effective	N/A	1. Depending on nebulous "Policy"
F5. Defenses for Signal Injection Are Missing	Charging profiles	1. No binding for identification with functionality; 2. Volatile context hash

Table IV: TCA evaluation using findings based on our systematization – While TCA has successfully pinpointed some urgent needs to solve the USB security problem, the design flaws and limitations render its goals in vain.

Although flawed, TCA is a promising start, since authentication is a necessary prerequisite to providing further security guarantees. We sketch several future research thrusts covering all findings from systematization to solve the USB security problem.

- **Solution Integration** Because most existing USB defense solutions focus on a single layer, it is natural to investigate how to combine different solutions covering multiple layers. For instance, ProvUSB+GoodUSB+FirmUSB expands the defense from Human Layer to Transport Layer, defeating most software-based attacks. USBFirewall is also a missing front-end for USBFILTER. Together, they produce a powerful USB packet firewall to control USB device behaviors while defending against exploitations from malformed packets. This will address our findings **F2** and **F4** the same time.
- **Type-C Authentication Products Evaluation** While we have shown design flaws of TCA, it is unlikely to see a new version of the spec in the near future, given that the spec has just been finalized. There is therefore an urgent need to evaluate the security of these new products, since real-world attacks might be the best push for a spec update. [3] It is also possible that vendor-specific implementations have considered those pitfalls in the spec, and offered some mitigations, which, once verified, provide convincing fixes to the spec. This will further improve the security guarantee of TCA on **F1** and **F5**.
- **Bi-directional Authentication and Mutual Authentication** While the trust anchor for USB hosts is missing in TCA, a short-term fix is to leverage the trusted hardware available on the host, such as TPM, and implement a host authentication protocol like Kells and ProvUSB. The possibility of doing bidirectional authentication also opens a door to mutual authentication, where the USB host and peripheral authenticate each other. This will also be the ultimate solution for **F3**.
- **Legacy Device Authentication** To authenticate legacy devices, two techniques are promising, and solve the problem in different ways. USB host fingerprinting has shown the possibility to fingerprint host machines via the USB interface using machine learning algorithms. The same idea could be applied to USB device fingerprinting, although with pitfalls of building a robust machine learning system in an adversarial environment. FirmUSB is able to understand the USB device firmware behavior, and thus provides a stronger security guarantee than fingerprinting when the firmware is available. We believe the combination of fingerprinting and firmware verification can mitigate most attacks from legacy devices. Together with TCA, this will provide a complete solution addressing issues in **F1**.
- **Policy Instantiation** Although security policies have been designed and used in existing solutions such as USBFILTER and Cinch, we need a new policy design that is general enough to be adopted by most vendors and expressive enough to ease creating rich rules. The new design should enumerate a set of subject, object, and access primitives to provide an intuitive mediation abstraction, define a common data marshaling format (e.g., XML, JSON) through which policies can be shared between deployments. It should also describe best practices for policy design, including how policies can preserve security in the presence of legacy devices. This will not only concrete TCA on **F2** and **F4**, but also promote USB security as part of systems security solutions, such as SELinux.

[3]Just like how Type-C Authentication spec was triggered by bad-quality charging cables.

VII. Conclusion

USB, after three generations and a recent connector change, remains woefully problematic. In this work, we present a structured methodology for reasoning about the nature of USB attacks and defenses. In so doing, we discover that these vulnerabilities harken back to the core trust-by-default principle of the USB specification, and identify design tradeoffs and principles that inform the properties of proposed defensive solutions. Finally, we formally verify the new USB Type-C Authentication specification, and uncover design flaws and implementation pitfalls. We conclude with future research directions. It is our intent that this systematization will guide future research efforts and ultimately improve the security of USB ecosystem.

References

[1] BlackShades malware hijacked half a million computers, FBI says. http://www.cbsnews.com/news/blackshades-malware-hijacked-half-a-million-computers-fbi-says/. Accessed: 2016-11-10.

[2] Cve-2013-1285: Windows usb descriptor vulnerability. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2013-1285.

[3] CVE-2013-1286: Windows USB Descriptor Vulnerability. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2013-1286.

[4] CVE-2013-1287: Windows USB Descriptor Vulnerability. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2013-1287.

[5] TURNIPSCHOOL - NSA playset. http://www.nsaplayset.org/turnipschool.

[6] COTTONMOUTH-I. https://nsa.gov1.info/dni/nsa-ant-catalog/usb/index.html#COTTONMOUTH-I, 2008.

[7] COTTONMOUTH-II. https://nsa.gov1.info/dni/nsa-ant-catalog/usb/index.html#COTTONMOUTH-II, 2008.

[8] Social Engineering a USB Drive. https://www.cmu.edu/iso/aware/be-aware/usb.html, 2016.
[9] AIA Vision Online. USB3 Vision Specification. http://www.visiononline.org/vision-standards-details.cfm?id=167&type=11, 2008.
[10] Alex Washburn. Snowden Smuggled Documents From NSA on a Thumb Drive. https://www.wired.com/2013/06/snowden-thumb-drive/, 2013.
[11] Alexandru Cornea. linux-usb: [PROBLEM] usb hub malformed packets causes null pointer dereference. http://marc.info/?l=linux-usb&m=144717111312054&w=2, 2016.
[12] ALLOYSEED. GIM Answer Monitor USB Charging Data Cable GPS Locator. https://www.aliexpress.com/item/1m-GPS-Positioning-Pick-up-Line-Tracker-Remote-Tracking-Cable-\GIM-Answer-Monitor-USB-Charging-Data/32813314360.html?trace=msiteDetail2pcDetail, 2017.
[13] S. Angel, R. S. Wahby, M. Howald, J. B. Leners, M. Spilo, et al. Defending against Malicious Peripherals with Cinch. In *USENIX Security Symposium*, Aug. 2016.
[14] Apple, Hewlett-Packard, Intel, Microsoft, Renesas, STMicroelectronics, and T. Instruments. Universal Serial Bus 3.2 Specification, Revision 1.0, September 2017.
[15] J. Bang, B. Yoo, and S. Lee. Secure usb bypassing tool. *digital investigation*, 7:S114–S120, 2010.
[16] D. Barrall and D. Dewey. Plug and Root, the USB Key to the Kingdom. In *Black Hat Briefings*, 2005.
[17] A. Bates, R. Leonard, H. Pruse, D. Lowd, and K. R. B. Butler. Leveraging USB to Establish Host Identity Using Commodity Devices. In *Proceedings of the 21st ISOC Network and Distributed System Security Symposium (NDSS'14)*, San Diego, CA, USA, Feb. 2014.
[18] A. Bates, D. Tian, K. R. Butler, and T. Moyer. Trustworthy Whole-System Provenance for the Linux Kernel. In *Proceedings of 24th USENIX Security Symposium on USENIX Security Symposium*, Aug. 2015.
[19] Benson Leung. Surjtech's 3M USB A-to-C cable completely violates the USB spec. Seriously damaged my laptop. https://www.amazon.com/review/R2XDBFUD9CTN2R/ref=cm_cr_rdp_perm, 2016.
[20] Binyamin Sharet. linux-usb: Gadgetfs - adding support for delegation of setup requests. http://marc.info/?l=linux-usb&m=147102748419146&w=2, 2016.
[21] B. Blanchet, V. Cheval, X. Allamigeon, and B. Smyth. Proverif: Cryptographic protocol verifier in the formal model. *URL http://prosecco. gforge. inria. fr/personal/bblanche/proverif*, 2010.
[22] T. Bosschert. Battling anti-forensics: Beating the u3 stick. *Journal of Digital Forensic Practice*, 1(4):265–273, 2007.
[23] S. Bratus, T. Goodspeed, P. C. Johnson, S. W. Smith, and R. Speers. Perimeter-crossing buses: a new attack surface for embedded systems. In *Proceedings of the 7th Workshop on Embedded Systems Security (WESS 2012), A Workshop of the Embedded Systems Week (ESWEEK)*, 2012.
[24] M. Brocker and S. Checkoway. iseeyou: Disabling the macbook webcam indicator led. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 337–352, 2014.
[25] K. Butler, S. McLaughlin, and P. McDaniel. Kells: A Protection Framework for Portable Data. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 231–240, New York, NY, USA, 2010. ACM.
[26] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 400–409, New York, NY, USA, 2009. ACM.
[27] A. Caudill and B. Wilson. Phison 2251-03 (2303) Custom Firmware & Existing Firmware Patches (BadUSB). *GitHub*, 26, Sept. 2014.
[28] Center for Development of Security Excellence. Security posters. http://www.cdse.edu/resources/posters.html.
[29] Common Vulnerabilities and Exposures. CVE-2006-2147. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2147, 2006.
[30] Common Vulnerabilities and Exposures. CVE-2010-2568. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568, 2010.
[31] Compaq, D. E. Corporation, I. P. Company, Intel, Microsoft, NEC, and N. Telecom. Universal Serial Bus Specification, Revision 1.0, January 1996.
[32] Compaq, Intel, Microsoft, and NEC. Universal Serial Bus Specification, Revision 1.1, September 1998.
[33] Comptia. Cyber secure: a look at employee cybersecurity habits in the workplace. https://www.comptia.org/resources/cyber-secure-a-look-at-employee-cybersecurity-habits-in-the-workplace, 2015.
[34] S. Corporation and I. Corporation. Universal Serial Bus Common Class Specification, Revision 1.0, December 1997.
[35] A. Cunningham. https://arstechnica.com/gadgets/2014/08/a-brief-history-of-usb-what-it-replaced-and-what-has-failed-to-replace-it/.
[36] Dan Patterson. How to build an external GPU for 4K video editing, VR, and gaming. http://www.techrepublic.com/article/how-to-build-an-external-gpu-for-4k-video-editing-vr-and-gaming/, 2016.
[37] Darren Pauli. Secret defence documents lost to foreign intelligence. http://www.itnews.com.au/news/secret-defence-documents-lost-to-foreign-intelligence-278961, 2011.
[38] A. Davis. Lessons learned from 50 bugs: Common usb driver vulnerabilities. Technical report, technical report, NCC Group, 2013.
[39] A. Davis. Revealing Embedded Fingerprints: Deriving Intelligence from USB Stack Interactions. In *Blackhat USA*, July 2013.
[40] S. A. Diwan, S. Perumal, and A. J. Fatah. Complete security package for USB thumb drive. *Computer Engineering and Intelligent Systems*, 5(8):30–37, 2014.
[41] Dominic Spill. USBProxy. https://github.com/dominicgs/USBProxy, 2014.
[42] Elizabeth Weise. A hacker's best friend is a nice employee. http://www.usatoday.com/story/tech/news/2016/08/15/hacker-social-engineering-defcon-black-hat/88621412/, 2016.
[43] Ellisys. USB Explorer 200 USB 2.0 Protocol Analyzer. http://www.ellisys.com/products/usbex200/index.php, 2013.
[44] Falliere, Nicolas and Murchu, Liam O and Chien, Eric. W32.Stuxnet Dossier. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, 2011.
[45] N. Fitzgibbon and M. Wood. Conficker. C: A technical analysis. *SophosLabs, Sophon Inc*, 2009.
[46] U. I. Forum. Universal Serial Bus Device Class Definition for Content Security Devices, Release 2.0, June 2012.
[47] U. I. Forum. Universal Serial Bus Power Delivery Specification, Revision 2.0 V1.2, March 2016.
[48] D. Genkin, I. Pipman, and E. Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs. *Journal of Cryptographic Engineering*, 5(2):95–112, 2015.
[49] GoodFET. Facedancer21. http://goodfet.sourceforge.net/hardware/facedancer21/, 2016.
[50] Google. Found Linux kernel USB bugs. https://github.com/google/syzkaller/blob/master/docs/linux/found_bugs_usb.md, 2017.
[51] M. Guri, M. Monitz, and Y. Elovici. Usbee: Air-gap covert-channel via electromagnetic emission from usb. *arXiv preprint arXiv:1608.08397*, 2016.
[52] Hak5. Episode 709: USB Rubber Ducky Part 1. http://hak5.org/episodes/episode-709, 2013.
[53] Hak5. USB Rubber Ducky Payloads. https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads, 2013.
[54] G. Hernandez, F. Fowze, D. J. Tian, T. Yavuz, and K. Butler. FirmUSB: Vetting USB Device Firmware using Domain Informed Symbolic Execution. In *24th ACM Conference on Computer and Communications Security (CCS'17)*, Dallas, USA, 2017.
[55] Hewlett-Packard, Intel, Microsoft, NEC, S.-N. Wireless, and T. Instruments. Universal Serial Bus 3.0 Specification, Revision 2.0, November 2008.
[56] Hewlett-Packard, Intel, Microsoft, Renesas, ST-Ericsson, and T. Instruments. Universal Serial Bus 3.1 Specification, Revision 1.0, July 2013.
[57] Hewlett-Packard, Intel, Microsoft, Renesas, STMicroelectronics, and T. Instruments. Universal Serial Bus Type-C Cable and Connector Specification, Revision 1.1, April 2015.
[58] H. J. Highland. The brain virus: fact and fantasy. *Computers & Security*, 7(4):367–370, 1988.
[59] Imation. IronKey Secure USB Devices Protect Against BadUSB Malware. http://www.ironkey.com/en-US/solutions/protect-against-badusb.html, 2014.
[60] IronKey. Ironkey. http://www.ironkey.com/en-US/resources/, 2013.
[61] J. R. Jacobs. Measuring the effectiveness of the USB flash drive as a vector for social engineering attacks on commercial and residential computer systems. Master's thesis, Embry-Riddle Aeronautical University, 2011.

[62] M. Jodeit and M. Johns. USB Device Drivers: A Stepping Stone into your Kernel. DEEPSEC, 2009.
[63] P. Johnson, S. Bratus, and S. Smith. Protecting Against Malicious Bits On the Wire: Automatically Generating a USB Protocol Parser for a Production Kernel. In *Proceedings of the 33th Annual Computer Security Applications Conference*, ACSAC '17, 2017.
[64] P. C. Johnson. *TOWARDS A VERIFIED COMPLEX PROTOCOL STACK IN A PRODUCTION KERNEL: METHODOLOGY AND DEMONSTRATION*. PhD thesis, DARTMOUTH COLLEGE Hanover, New Hampshire, 2016.
[65] S. N. Jones, C. R. Strong, D. D. E. Long, and E. L. Miller. Tracking Emigrant Data via Transient Provenance. In *3rd Workshop on the Theory and Practice of Provenance*, TAPP'11, June 2011.
[66] S. Kamkar. USBdriveby. http://samy.pl/usbdriveby/, 2014.
[67] Kanguru Solutions. Secure Encrypted USB Flash Drives. https://www.kanguru.com/.
[68] KeeLog. Hardware Keylogger. https://www.keelog.com/, 2016.
[69] Kevin Poulsen and Kim Zetter. U.S. Intelligence Analyst Arrested in Wikileaks Video Probe. http://www.wired.com/2010/06/leak/, 2010.
[70] P. Kumaraguru, Y. Rhee, S. Sheng, S. Hasan, A. Acquisti, L. F. Cranor, and J. Hong. Getting users to pay attention to anti-phishing education: evaluation of retention and transfer. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 70–81. ACM, 2007.
[71] J. Larimer. Beyond autorun: Exploiting vulnerabilities with removable storage. In *Blackhat DC*, 2011.
[72] L. Letaw, J. Pletcher, and K. Butler. Host Identification via USB Fingerprinting. *Systematic Approaches to Digital Forensic Engineering (SADFE), 2011 IEEE Sixth International Workshop on*, pages 1–9, May 2011.
[73] Y. Li, J. M. McCune, and A. Perrig. VIPER: Verifying the Integrity of PERipherals' Firmware. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 3–16. ACM, 2011.
[74] V. C. Luo. Tracing USB Device artefacts on Windows XP operating system for forensic purpose. In *5th Australian Digital Forensics Conf*, 2007.
[75] Mathew J. Schwartz. How USB Sticks Cause Data Breach, Malware Woes. http://www.pcworld.com/article/237600/companies_lose_2_5_million_from_missing_memory_sticks_study_says.html, 2011.
[76] C. McGarry. New macbook pros reportedly going all in with usb-c. http://www.macworld.com/article/3132395/hardware/new-macbook-pros-reportedly-going-all-in-with-usb-c.html, Oct 2016.
[77] mich. Inside a low budget consumer hardware espionage implant. https://ha.cking.ch/s8_data_line_locator/, 2017.
[78] Microsoft. Security intelligence report. https://www.microsoft.com/security/sir/default.aspx, 2015.
[79] Microsoft Windows Embedded 8.1 Industry. Usb filter (industry 8.1). https://msdn.microsoft.com/en-us/library/dn449350(v=winembedded.82).aspx, 2014.
[80] MWR Labs. Usb fuzzing for the masses. https://labs.mwrinfosecurity.com/blog/usb-fuzzing-for-the-masses/, July 2011.
[81] M. Neugschwandtner, A. Beitler, and A. Kurmus. A transparent defense against usb eavesdropping attacks. In *Proceedings of the 9th European Workshop on System Security*, EuroSec '16, pages 6:1–6:6, New York, NY, USA, 2016. ACM.
[82] K. Nohl. BadUSB Exposure: Hubs. https://opensource.srlabs.de/projects/badusb/wiki/Hubs, November 2014.
[83] K. Nohl and J. Lehl. BadUSB – On Accessories That Turn Evil. In *Blackhat USA*, Aug. 2014.
[84] OLEA Kiosks, Inc. Malware Scrubbing Cyber Security Kiosk. http://www.olea.com/product/cyber-security-kiosk/, 2015.
[85] P. Oliveira, Jr. FBI can turn on your web cam, and you'd never know it. http://nypost.com/2013/12/08/fbi-can-turn-on-your-web-cam/, 8 Dec. 2013. Accessed: 2016-11-10.
[86] OPSWAT. Metascan. https://www.opswat.com/products/metascan, 2013.
[87] D. Oswald, B. Richter, and C. Paar. Side-channel attacks on the yubikey 2 one-time password generator. In *International Workshop on Recent Advances in Intrusion Detection*, pages 204–222. Springer, 2013.
[88] J. Patrick-Evans, L. Cavallaro, and J. Kinder. POTUS: Probing off-the-shelf USB drivers with symbolic fault injection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, 2017. USENIX Association.
[89] Paul Sewers. US Govt. plant USB sticks in security study, 60the bait. http://thenextweb.com/insider/2011/06/28/us-govt-plant-usb-sticks-in-security-study-60-of-subjects-take-the-bait/, 2011.
[90] D. Pham, M. Halgamuge, A. Syed, and P. Mendis. Optimizing windows security features to block malware and hack tools on USB storage devices. In *Progress in electromagnetics research symposium*, 2010.
[91] S. Poeplau and J. Gassen. A Honeypot for Arbitrary Malware on USB Storage Devices. In *7th International Conference on Risk and Security of Internet and Systems*, CRiSIS '12, Oct. 2012.
[92] P. Porras, H. Saidi, and V. Yegneswaran. Conficker c analysis. *SRI International*, 2009.
[93] R. S. Portnoff, L. N. Lee, S. Egelman, P. Mishra, D. Leung, and D. Wagner. Somebody's watching me?: Assessing the effectiveness of webcam indicator lights. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1649–1658, New York, NY, USA, 2015. ACM.
[94] Renesas. Renesas Electronics Delivers R9J02G012 Controller That Enables Device-to-Device Authentication in Support of Safer USB Power Delivery Ecosystem. https://www.renesas.com/en-hq/about/press-center/news/2017/news20170530.html, 2017.
[95] D. Rich. Authentication in transient storage device attachments. *Computer*, 40(4):102–104, April 2007.
[96] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *2012 IEEE Symposium on Security and Privacy*, pages 224–238, May 2012.
[97] Rosie Hall. linux-usb: USB vulnerability. http://marc.info/?l=linux-usb&m=146972356605600&w=2, 2016.
[98] J. Saxe, D. Mentis, and C. Greamo. Mining web technical discussions to identify malware capabilities. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 1–5. ieeexplore.ieee.org, July 2013.
[99] N. Schear, C. Kintana, Q. Zhang, and A. Vahdat. Glavlit: Preventing Exfiltration at Wire Speed. In *Proceedings of the ACM Workshop on Hot Topics in Networks*, HotNets, November 2006.
[100] S. Schumilo, R. Spenneberg, and H. Schwartke. Don't trust your USB! How to find bugs in USB device drivers. In *Blackhat Europe*, Oct. 2014.
[101] Sergey Bratus and Travis Goodspeed. Facedancer USB: Exploiting the Magic School Bus. https://recon.cx/2012/schedule/events/237.en.html, 2012.
[102] G. Shah, A. Molina, and M. Blaze. Keyboards and Covert Channels. In *Proceedings of the 2006 USENIX Security Symposium*, pages 59–75, Aug. 2006.
[103] S. Sheng, M. Holbrook, P. Kumaraguru, L. F. Cranor, and J. Downs. Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 373–382. ACM, 2010.
[104] S. Shin and G. Gu. Conficker and Beyond: A Large-scale Empirical Study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 151–160, New York, NY, USA, 2010. ACM.
[105] Siliconch Systems. USB Type-C Authentication IP. http://www.siliconch.com/authentication.html, 2017.
[106] K. Sridhar, S. Prasad, L. Punitha, and S. Karunakaran. Emi issues of universal serial bus and solutions. In *Electromagnetic Interference and Compatibility, 2003. INCEMIC 2003. 8th International Conference on*, pages 97–100. IEEE, 2003.
[107] S. Stasiukonis. Social engineering, the USB way. *Dark Reading*.
[108] Y. Su, D. Genkin, D. Ranasinghe, and Y. Yarom. USB snooping made easy: Crosstalk leakage attacks on USB hubs. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1145–1161, Vancouver, BC, 2017. USENIX Association.
[109] P. Szor. Duqu–threat research and analysis. *McAfee Labs*, 2011.
[110] Tal Ater. Chrome Bugs Allow Sites to Listen to Your Private Conversations. https://www.talater.com/chrome-is-listening/, 2014.
[111] The USB Device Working Group. USB Class Codes. http://www.usb.org/developers/defined_class, 2015.
[112] D. J. Tian, A. Bates, and K. R. B. Butler. Defending Against Malicious USB Firmware with GoodUSB. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC '15, 2015.

[113] D. J. Tian, A. Bates, K. R. B. Butler, and R. Rangaswami. Provusb: Block-level provenance-based data protection for usb storage devices. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, Oct 2016. ACM.

[114] D. J. Tian, N. Scaife, A. Bates, K. R. B. Butler, and P. Traynor. Making usb great again with usbfilter. In *25th USENIX Security Symposium (USENIX Security 16)*, Washington, D.C., 2016. USENIX Association.

[115] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey. Users Really Do Plug in USB Drives They Find. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P '16)*, San Jose, California, USA, May 2016.

[116] USB 3.0 Promoter Group. Universal Serial Bus Type-C Authentication Specification, Revision 1.0, March 2016.

[117] USB Implementers Forum. USB-IF statement regarding USB security. http://www.usb.org/press/USB-IF_Statement_on_USB_Security_FINAL.pdf, aug 2014.

[118] USB Implementers Forum, Inc. USB Mass Storage Class CBI Transport. http://www.usb.org/developers/docs/devclass_docs/usb_msc_cbi_1.1.pdf, 2003.

[119] USB Implementers Forum, Inc. USB Mass Storage Class Specification Overview. http://www.usb.org/developers/docs/devclass_docs/Mass_Storage_Specification_Overview_v1.4_2-19-2010.pdf, 2010.

[120] USBKiller. Usbkiller. https://www.usbkill.com/, 2016.

[121] R. D. Vega. USB Attacks: Fun with Plug and 0wn. https://labs.mwrinfosecurity.com/assets/135/original/mwri_t2-usb-fun-with-plug-and-0wn_2009-10-29.pdf, October 2009.

[122] D. Wagenaar, D. Pavlov, and S. Yannick. USB baiting. *Universite van Amserdam*, 2011.

[123] J. Walter. "Flame Attacks": Briefing and Indicators of Compromise. *McAfee Labs Report*, May 2012.

[124] P. Walters. The risks of using portable devices, 2012.

[125] Z. Wang and A. Stavrou. Exploiting Smart-phone USB Connectivity for Fun and Profit. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 357–366, New York, NY, USA, 2010. ACM.

[126] B. Yang, D. Feng, Y. Qin, Y. Zhang, and W. Wang. TMSUI: A Trust Management Scheme of USB Storage Devices for Industrial Control Systems. Cryptology ePrint Archive, Report 2015/022, 2015. http://eprint.iacr.org/.

[127] K. Zetter. Meet 'flame,' the massive spy malware infiltrating iranian computers. *Wired*, 28 May 2012.

[128] S. Zhioua. The middle east under malware attack dissecting cyber weapons. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 11–16. ieeexplore.ieee.org, July 2013.