USB Proxy

Robbie Dumitru, Mark Beaumont, Bradley Hopkins, and Simon Windows
Defence Science and Technology Group
Adelaide, Australia
{robbie.dumitru,mark.beaumont,bradley.hopkins1,simon.windows}@defence.gov.au

ABSTRACT

Universal Serial Bus (USB) is the de facto standard for computer-peripheral interconnect, however over the course of its development little consideration has been given to security. This has led to multiple examples demonstrating how USB can be leveraged to undermine the security of computer systems across different layers of its interface. User-friendly, holistic security solutions have proven challenging to develop, especially given they must be retrofitted to existing systems as add-on elements.

We have built a prototype USB Proxy device which brings together several protective functions for securing USB. The device serves as a lightweight mediator for USB connections. With minimal cost to performance, it enforces canonicalisation of data transferred between a device and host, allowing only known-good behaviour. The connection separation prevents malicious behaviour, such as bus-sniffing exploits. It also anonymises both ends of a connection, providing confidentiality against profiling from either side.

ACM Reference Format:
Robbie Dumitru, Mark Beaumont, Bradley Hopkins, and Simon Windows. 2023. USB Proxy. In *2023 Australasian Computer Science Week (ACSW 2023), January 30-February 3, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3579375.3579390

1 INTRODUCTION

USB is the most prevalent means of connecting computers with peripheral devices. Development of the USB standard has consistently been driven toward ease-of-use and low-cost implementation which has helped drive its popularity over the past two decades, however, these characteristics are often at odds with secure design. By design, the technology assumes mutual trust between the host (computing system) and connecting devices. In the absence of such trust, USB presents a very compelling attack vector because of its lack of inbuilt security features, the level of system access it can offer, and its functional versatility. Many prior works have demonstrated attacks on the USB ecosystem via compromised devices, such as USB sticks masquerading as keyboards to send commands [23], hubs on a device's connection path monitoring and manipulating traffic [22], and even off-path devices monitoring [28, 31] and manipulating [16] the traffic of other devices.

With the USB Implementers Forum[1] stating that "consumers should only grant trusted sources with access to their USB devices" [34], USB's security model relies on limiting port access to trusted devices rather than tried and tested techniques such as encryption, authentication, and access control. This is insufficient where critical systems are concerned, particularly given threats posed by supply chain compromise, surreptitious device planting, and social engineering approaches [3, 33].

Several protective measures have been proposed to enable safe interaction with potentially untrustworthy components across a USB connection, including firewall-style device authorisation policies, physical bus separation, and cryptographic overlay protocols. Common shortcomings among many of these defence implementations is that they either severely impact performance, or they only prevent certain attacks and at certain layers. Tian et al. [32] emphasise that many of the most effective attacks work across multiple layers of the USB protocol stack, highlighting the need for holistic security solutions.

Our Contribution. In this work we present a protective hardware prototype device called the *USB Proxy* which interposes the connection between a host and peripheral USB device. The Proxy uniquely consolidates multiple defensive functions of device and data filtering, interface canonicalisation, physical bus separation, and double-blind anonymity. It facilitates trustworthy virtual USB connections as an active mediator by establishing two separate outward host- and device-side USB connections. Connection through the Proxy incurs minimal performance costs when compared to a direct connection. Only enumeration time is affected.

When connecting a device through the Proxy it will first determine, depending on the configured filter policy, whether to allow a virtual connection to the host. Physical separation from the host bus means the Proxy will prevent any exploits based on bus or traffic sniffing and manipulation. This includes all known mechanisms of host fingerprinting using devices. Furthermore, the Proxy will mask the connecting device's identity by presenting user-configured identifiers toward the host. This double-blind anonymity between host- and device-side means the USB interface cannot be exploited to profile either side in any intelligence gathering activities.

In this paper we discuss related work pertaining to USB security (Section 2). We detail the architecture of the Proxy (Section 3), our prototype implementation (Section 4), and its operation which we evaluate (Section 5).

2 BACKGROUND

USB was first released in 1996 to simplify the use of peripherals that hitherto required different interfaces for each device type. Aside from reducing the number of port types computers would have to ship with, the other major simplification it introduced was that

[1]The organisation responsible for development of the USB standard

ACSW 2023, January 30-February 3, 2023, Melbourne, VIC, Australia
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0005-7/23/01…$15.00
https://doi.org/10.1145/3579375.3579390

peripherals would automatically enumerate (self-configure) upon being plugged in, referred to as 'plug-and-play'.

2.1 USB Basics

USB is a host-managed shared bus. The shared bus network is structured in a tree topology with a host (computer) at the root, hubs that branch off to extend the number of attachment points, and devices at the leafs. All transactions are scheduled and initiated by the host, one at a time. When a device connects to a host over USB the first action is for the host to enumerate the device. During enumeration the host requests the device to send its descriptor set. This is self-reported and non-authenticated information which the device provides. Based on the descriptors, the host identifies the device and loads the appropriate software drivers.

2.2 USB-based Attacks

Given its prevalence among PCs, IoT devices, and other embedded systems, USB is an appealing avenue of exploitation for malicious actors seeking to compromise computer systems. USB's lack of access control mechanisms and encryption is particularly problematic as it makes attack vectors simple to develop and effective, whereas countermeasures are typically costly and complicated. Here we review several classes of USB attacks and defences, for more comprehensive summaries see [25, 26, 29, 30, 32] and references therein.

Electrical Attacks. Commercially available devices like the *USB Killer* [5] can permanently incapacitate host computers and devices attached to them by carrying out power surge electrical attacks when plugged in. Such devices are equipped with capacitors that they charge up using the USB power supply, to then discharge a destructive high voltage direct current over the USB data lines.

Host Exploits. This attack class is labelled as such because it uses connection through regular USB devices as an attack vector for exploiting vulnerabilities in host systems. Halderman et al. [20] exploit the lingering persistence of RAM contents after loss of power by performing a 'Cold-boot' from a USB drive and dumping the residual memory, recovering user login information and encryption keys. Vulnerabilities like buffer overflows [1] in USB drivers can facilitate the execution of malicious code on host systems. A large body of work [13, 18, 19] is dedicated to fuzzing the USB driver stack for vulnerabilities. Similarly, compromise of the driver update procedure of an OS can enable registration of drivers containing malicious executables on any system that runs the OS.

Device Masquerading. USB device firmware can be programmed (or re-programmed) to emulate the operation of certain devices. A popular exploit method uses seemingly innocuous devices, typically USB sticks, to emulate HID (human interface device) keyboards [15, 17, 21, 23]. These emulators feed keystroke inputs to achieve certain objectives such as establishing remote shells, accessing malicious websites, overriding DNS settings [21], or data exfiltration [15]. Keystroke sequence payloads can be pre-loaded on-chip [23] or remotely fed [17]. Devices programmed to emulate HIDs can also deceive victim users once plugged in by functioning as what they physically appear to be. For example, a USB stick can work as a compound device which establishes both a mass storage and a HID keyboard interface over the same USB connection. Alternatively, it might initially connect just as mass storage before disconnecting and reconnecting as a HID keyboard.

Bus Sniffing. USB is a host-managed shared bus whose traffic is not encrypted. In USB (version 2.0 and older) all downstream (host-to-device) transmissions are broadcast on the bus making it possible for connected devices to monitor communication toward adjacent devices [28]. [16] demonstrates that devices can listen to downstream probes addressed to other devices and then inject upstream data on their behalf. Upstream transmissions are only meant to reach the host, however these signals have been demonstrated to leak between ports [31] due to crosstalk leakage effects.

2.3 Defences

A consequence of the USB standard being geared towards low-cost implementation is that the resultant technology is not particularly robust. Electrical attacks and upstream crosstalk leakage can be mitigated by electrical isolation of USB ports, but this is a costly measure. The trade-off between security and usability is well-exemplified by USB. Angel et al. [12] implement encryption and authentication over USB at an 80% cost to performance by measure of throughput. Device authorisation policies [6, 10, 27] can be used to allow or block interaction with certain devices. Although, in most cases these implementations fingerprint devices based on unauthenticated device-provided descriptors.

Commercial Protective Solutions. Physical USB port blockers [7, 8] are commercially available tools for locking down USB computer ports and making them unusable. *HighSecLabs*' suite of USB protection device products [4] features mechanical port locks and USB filters. One of the two filters advertised blocks all USB traffic except for keyboards and mice, the other allows connections based on device descriptor fields. The *USB Sentry* [9] is a secure USB hub used in a similar vein by blocking all other ports. It prevents some protocol violations, bus sniffing, and tricks commonly used in device emulation exploits. *Globotron* produce the *USG v1.0 Hardware Firewall* [11] and the *Armadillo Hardware Firewall USB 2.0* [2]. Both firewalls only enable connection of mass storage, mice and keyboard devices, all with limited functionality. They detect and block malicious input from mice and keyboards, as would be provided by HID emulators, based on input types and timing.

3 ARCHITECTURE

Our proposed USB Proxy comprises two distinct USB entities: a *Proxy-device* (host-facing) and *Proxy-host* (device-facing). Data is selectively transferred between the two to establish a virtual USB connection between an external host and an external peripheral. The Proxy has full control over the connection.

Operation. Power is supplied to the USB Proxy from the external host (e.g., host PC). Upon connection to an external device (e.g., keyboard), the Proxy-host enumerates the device, obtaining its descriptor set and bringing it into a connected state (only with respect to the Proxy-host). From the descriptor set, the Proxy determines if the device is authorised for connection. If the device is not authorised no further action is taken i.e., no connection to the external host is established, therefore no enumeration to the host occurs. The Proxy can be configured to maintain a combination of explicitly trusted or non-trusted descriptor fields by which to filter devices that are enabled a connection toward the host, thereby implementing its own device authorisation policy.

Should the enumerated device be authorised, then on its behalf the Proxy-device will emulate an equivalent *generic* device of the same type toward the external host. From this the host establishes a connection with the Proxy-device across interfaces defined by the descriptor set it provides. Certain fields in the generic descriptor set are modifiable in configuration, namely the string descriptors and *idVendor*, *idProduct*, and *bcdDevice* byte pairs. Operating systems typically use the globally unique combinations of *idVendor* and *idProduct* (often referred to as VID and PID, respectively) fields to identify USB devices. *bcdDevice* is the device version number. String descriptors are optional human readable descriptors.

Virtual connection with the external device is facilitated by passing data between Proxy-host and Proxy-device. By emulating a generic device on the real device's behalf, the Proxy canonicalises the data flows between external host and device because it strictly defines the interfaces established. Data in transit can be filtered or altered according to configured policy. Capture and storage of the data is also possible for post-use analysis.

4 PROTOTYPE

We have developed a prototype implementation of the USB Proxy based on a programmable embedded USB controller (FTDI VNC2 [14]), that has both a host USB interface and a separate device USB interface. The controller is attached to two USB connectors, LEDs, a push button switch, and an EEPROM. See Figure 1. All components are housed on a board roughly the size of a thumb drive.

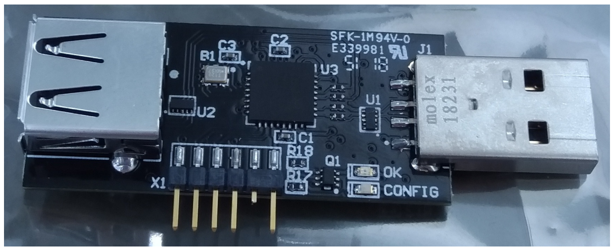

Figure 1: USB Proxy prototype on custom board

Figure 2 illustrates the data flows within the Proxy prototype. The connections toward external entities are at the USB connectors represented by *PC* and *Mouse/Keyboard* labels.

The FTDI VNC2 firmware was developed with separate software modules that control the host and device interfaces. Enumeration and other control exchanges are isolated to occur separately with both external entities. The host interface software (*proxy-host*) connects to peripheral devices (e.g., keyboard) that are attached. The device software (*Proxy-device*) establishes a connection, as a peripheral, to a host computer. Additional software then implements the actual proxy functionality, passing only data between the two interfaces through use of a shared memory.

The initial prototype supports keyboard and mouse peripheral devices. When a peripheral device is plugged in, *only* if it is a keyboard or mouse, it is enumerated by the *Proxy-host* and a connection is established. Then, the *Proxy-device* firmware enumerates with the host, and only does so using the boot protocol interface and presenting generic mouse or keyboard descriptor sets, depending on which type of device has been plugged in.

Mouse or keyboard input packets from the *Proxy-device* are the only data input that reach the host, as enforced by enumeration establishing an interface with the host which expects only this type of input. A size check is performed on input data sent to the host. With keyboards, output report bytes that convey PC state (Caps lock, Num lock, etc.) used for lighting up LEDs are sent as control transfers. This constitutes the only output data transferred.

Configuration. The device has two operational modes and by default is programmed to work as a USB Proxy. However, if the push button 'CONFIG' switch is pressed upon plugging the prototype into a host, the device powers on in *configuration mode*. In this mode it connects to the host as a serial device over USB.

Using this serial interface, host-based software can configure the Proxy device. For example, values for the modifiable descriptor fields (IDs, Strings) can be written to the Proxy and stored on its EEPROM for retention between power on, power off cycles. Thereon, in normal operation the Proxy assumes and presents the most recently configured identifying descriptors toward the host.

5 EVALUATION

To be effective, a USB Proxy must correctly enforce the required security functionality, all while facilitating a virtual USB connection with a negligible effect on performance. We validate this functionality by testing the prototype implementation described in Section 4.

Device Filtering. When plugging in non-authorised device types there is no interaction with the host. The Proxy handles compound devices which contain one of the allowable interfaces, enabling communication only through that interface. We verified that the entire USB descriptor set seen by hosts was consistent with only the Proxy supplied generic and configured device descriptors.

Interface Canonicalisation. Due to plug-and-play, there is potential for large discrepancy between the interface a device self-reports and what type of device the user thinks is plugged in. The Proxy prototype was verified to always report in the same boot protocol format for each of either keyboard or mouse devices - regardless of the underlying report structure of the peripheral plugged in.

Physical Separation. Downstream traffic is not visible toward devices connected through the Proxy that are capable of bus sniffing due to the separation of physical bus interfaces. Electrical isolation was outside of scope therefore our prototype may yet leave the USB connection susceptible to electrical attacks. Further measures such as current limiting or voltage spike clamping could be implemented as a means of handling this.

Anonymity. The *Proxy-device* presented to the host PC as a generic device, with USB IDs and strings that were unrelated to the actual keyboard or mouse plugged in - masking the identity of the connecting device. Probing from the host returned no identifying information from the actual peripheral device. Likewise the external peripheral device connected to the Proxy-host received no identifying information about the host PC (in characterisable enumeration request timings [24]) - masking host behaviour and identity.

Data Filtering. The Proxy performs no operations on the transferred data, therefore there are no measures preventing malformed

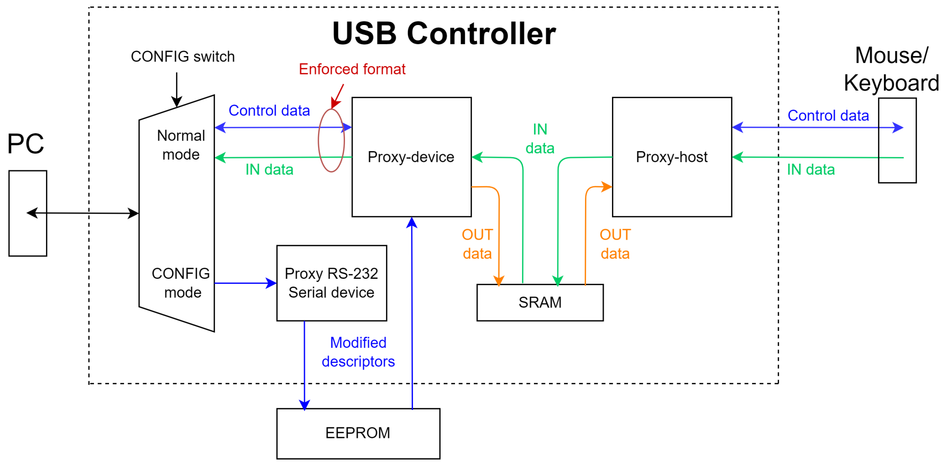

Figure 2: USB Proxy Prototype Data Flows

data being transferred to the host, however this data is only processed as mouse and keyboard input reports. Vulnerabilities in the host's mouse and keyboard drivers would still be exposed through use of the Proxy, with the exception of buffer overflow vulnerabilities since a size check is performed on the transferred data. Support for data monitoring can be incorporated.

Performance. The Proxy platform is capable of communicating at the same speeds as devices which it emulates, as such, no loss in bandwidth is incurred. There is a slightly increased delay due to input processing, however this delay is unnoticeable for human users. More significantly, due to the dual enumeration and authorisation checks, enumeration time is doubled.

6 CONCLUSION

A prototype USB Proxy was implemented that successfully implements the desired security functions of device/data filtering and physical separation as found among commercial solutions, along with additional security functions enforcing interface canonicalisation and anonymity. This proxy can relatively cheaply and simply increase security when using USB peripherals. The Proxy provides a configurable platform for mediating security for any type of USB peripheral. Future work includes expanding the proxy functionality to support other USB protocols, and implementing real-time data filtering techniques.

REFERENCES

[1] 2009. CVE-2009-4067: . https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4067
[2] 2022. Armadillo Hardware Firewall USB 2.0. https://globotron.nz/products/armadillo-hardware-usb-firewall
[3] 2022. Dead Drops. https://deaddrops.com/
[4] 2022. eLock USB Lockdown. http://www.highseclabs.com/products/?pid=92
[5] 2022. USB Kill. https://usbkill.com/
[6] 2022. USB-Lock-RP. https://www.usb-lock-rp.com/
[7] 2022. USB Physical Security. https://www.smartkeeperworld.com/#/usb-physicalsecurity
[8] 2022. USB Port Blocker. https://www.kensington.com/p/products/data-protection/usb-port-lock-security/usb-port-blocker-cable-guard-horizontal/
[9] 2022. USB Sentry. https://www.atcorp.com/products/usbsentry/
[10] 2022. USBGuard. https://usbguard.github.io/
[11] 2022. USG v1.0 Hardware Firewall. https://globotron.nz/products/usg-v1-0-hardware-usb-firewall
[12] Sebastian Angel et al. 2016. Defending against Malicious Peripherals with Cinch. In *USENIX Security Symposium 2016*.
[13] Sergey Bratus. 2012. Perimeter-Crossing Buses : a New Attack Surface for Embedded Systems. In *WESS 2012*.
[14] FTDI Chip. 2022. VNC2 - Vinculum-II Programmable USB 2.0 Host. https://www.ftdichip.com/old2020/Products/ICs/VNC2.htm
[15] John Clark, Sylvain Leblanc, and Scott Knight. 2011. Compromise through USB-based Hardware Trojan Horse device. *Future Generation Computer Systems* 27 (May 2011), 555–563.
[16] Robert Dumitru, Daniel Genkin, Andrew Wabnitz, and Yuval Yarom. 2023. The Impostor Among US(B): Off-Path Injection Attacks on USB Communications. In *USENIX Security Symposium 2023*.
[17] Monta Elkins. 2018. Universal RF USB Keyboard Emulation Device URFUKED. https://defcon.org/images/defcon-18/dc-18-presentations/Elkins/DEFCON-18-Elkins-Universal-RF-Keyboard.pdf
[18] Travis Goodspeed. 2022. Python USB device emulation. http://goodfet.sourceforge.net/hardware/facedancer21/
[19] Google. 2019. Found Linux kernel USB bugs. https://github.com/google/syzkaller/blob/master/docs/linux/found_bugs_usb.md
[20] J. Alex Halderman et al. 2009. Lest We Remember: Cold-Boot Attacks on Encryption Keys. 52, 5 (May 2009), 91–98. https://doi.org/10.1145/1506409.1506429
[21] Samy Kamkar. 2022. USBdriveby. http://samy.pl/usbdriveby/
[22] David Kierznowski. 2016. *BadUSB 2.0: USB man in the middle attacks*. Masters Thesis. Royal Holloway University of London.
[23] Darren Kitchen. 2022. USB Rubber Ducky Payloads. https://github.com/hak5/usbrubberducky-payloads
[24] Lara Letaw, Joe Pletcher, and Kevin Butler. 2011. Host Identification via USB Fingerprinting. *SADFE* (05 2011).
[25] Hao Liu, Riccardo Spolaor, Federico Turrin, Riccardo Bonafede, and Mauro Conti. 2021. USB powered devices: A survey of side-channel threats and countermeasures. *High-Confidence Computing* (2021).
[26] Mark Mamchenko and Alexey Sabanov. 2019. Exploring the Taxonomy of USB-Based Attacks. In *MLSD 2019*.
[27] Hessam Mohammadmoradi and Omprakash Gnawali. 2018. Making Whitelisting-Based Defense Work Against BadUSB. In *ICSDE 2018*.
[28] Matthias Neugschwandtner, Anton Beitler, and Anil Kurmus. 2016. A Transparent Defense Against USB Eavesdropping Attacks. In *EUROSEC 2016*.
[29] Nir Nissim, Ran Yahalom, and Yuval Elovici. 2017. USB-based attacks. *Comput. Secur.* 70 (2017), 675–688.
[30] Dung Vu Pham, Ali Syed, and Malka N. Halgamuge. 2011. Universal serial bus based software attacks and protection solutions. *Digit. Investig.* 7, 3–4 (2011), 172–184.
[31] Yang Su, Daniel Genkin, Damith Chinthana Ranasinghe, and Yuval Yarom. 2017. USB Snooping Made Easy: Crosstalk Leakage Attacks on USB Hubs. In *USENIX Security Symposium 2017*.
[32] Dave (Jing) Tian et al. 2018. SoK: "Plug & Pray" Today - Understanding USB Insecurity in Versions 1 Through C. In *IEEE Security and Privacy 2018*.
[33] Matthew Tischer et al. 2016. Users really do plug in usb drives they find. In *IEEE Security and Privacy 2016*.
[34] USB Implementers Forum. 2014. USB-IF Statement regarding USB security. https://www.usb.org/press/USB-IF_Statement_on_USB_Security_FINAL.pdf.