



T.R.

USKUDAR UNIVERSITY INSTITUTE OF SCIENCE

MASTER'S DEGREE PROGRAM OF CYBER SECURITY

MASTER'S THESIS

**DUCKTAPE AS A NEW COUNTERMEASURE AGAINST BAD
USB THREATS**

NOUREDDEEN ABDELFAHAT

Thesis Advisor

Asst. Prof. Ahmet Şenol

ISTANBUL-2023

T.R.
USKUDAR UNIVERSITY
INSTITUTE OF SCIENCE

MASTER'S DEGREE PROGRAM OF CYBER SECURITY

MASTER'S THESIS

**DUCKTAPE AS A NEW COUNTERMEASURE AGAINST BAD
USB THREATS**

NOUREDDEEN ABDELFATTAH

Thesis Advisor
Asst. Prof. Ahmet Şenol

ISTANBUL-2023

ABSTRACT

DUCKTAPE AS A NEW COUNTERMEASURE AGAINST BAD USB THREATS

This thesis investigates the vulnerabilities and risks associated with USB devices, particularly focusing on the prevalence of BadUSB attacks and the failure of systems to prevent them. The weaknesses in the USB protocol and the exploitation techniques employed by threat actors are explored. The research focuses on evaluating the efficacy of countermeasures in mitigating these threats while providing insights into existing solutions' limitations and potential enhancements. Through an in-depth analysis, the study identifies the behavioral characteristics of injection attacks and uncovers major vulnerabilities resulting from USB device misuse. Building upon these findings, a countermeasure called "DuckTape" is developed, emphasizing verification, isolation, logging, and privilege de-escalation techniques and optional elimination of unnecessary functions without compromising user experience. Experimentation is conducted to evaluate DuckTape's performance overhead and practicality, demonstrating its ability to minimize the threat surface without sacrificing usability. The thesis further discusses the limitations and highlights the possibilities for future improvements. Proposed DuckTape application enables a balanced approach to security and usability. This research expands the understanding of BadUSB threats and contributes valuable insights into mitigating these attacks. With a concise yet effective protection approach, DuckTape provides a versatile solution that can be easily implemented across different operating systems and security environments. The findings of this thesis underscore the importance of considering multiple layers of defense and serve as a catalyst for further advancements in protecting against BadUSB threats. Overall, this research contributes to enhancing the understanding of BadUSB threats and provides a valuable prevention mechanism for users and security-conscious environments, promoting a balance between security and usability in mitigating keystroke injection attacks.

Keywords: BadUSB, Keystroke Injection, Vulnerability Analysis, Countermeasures, DuckTape, Threat Surface Mitigation

FOREWORD

I extend my sincere gratitude to my advisor, for the invaluable guidance, support, and encouragement provided throughout the course of this research. Their valuable insights and expertise were instrumental in the successful completion of this thesis.

I am also grateful to my family and friends without whom the completion of this thesis would not have been possible, for their unwavering support and understanding throughout the entire research process.



TEXT of the OATH

I declare that this study is my own thesis study, that I have no unethical behavior at any stage from planning to writing, that I have obtained all the information in the thesis within academic and ethical rules, and that I resource all the information and comments that are not obtained through the thesis study.

04/08/2023

Nouredden Abdelfattah



CONTENTS

ABSTRACT.....	i
FOREWORD	ii
TEXT of the OATH	iii
CONTENTS	iv
INDEX OF TABLES	vi
INDEX OF FIGURES	vii
INDEX OF ABBREVIATIONS	viii
1. INTRODUCTION	1
1.1 Overview	1
1.2 Problem Statement	1
1.3 Objectives and Research Questions	1
1.4 Scope and Limitations	2
1.5 Significance.....	2
1.6 Thesis Organization	3
2. BACKGROUND	4
2.1. USB.....	4
2.2. BadUSB	6
2.3. Multiple Attacks on Multiple Layers	10
2.4. Threat Modelling.....	13
2.4.1. Insider Threat	13
2.4.2. Outsider Threat.....	14
2.4.3. Attack Attributes	16
3. RELATED WORK	19
3.2. Software Based Countermeasures Against USB Attacks.....	19
3.1. Hardware Based Solutions	30
3.3. Drawbacks.....	37

3.3.1. Observed Weaknesses.....	37
3.3.2. The Main Weakness: User Reliance	39
4. DUCKTAPE: A NEW COUNTERMEASURE AGAINST USB ATTACKS.....	41
4.1 Overview.....	42
4.2 Key Components and Implementation Details.....	43
4.3 Usability and Complexity:	45
4.4 Strengths and Advantages:	46
4.5 Limitations:	48
4.6 Testing.....	49
4.6.1. Testing Environment Setup:.....	49
4.6.1.1 Hardware Setup:.....	49
4.6.1.2 Software Setup:	50
4.6.2. Testing Methodology:	50
4.6.2.1 Test Scenarios and Use Cases:	50
4.6.2.2 Test Metrics and Measurements:	52
4.6.3 Testing Limitations:	54
4.6.4 Evaluation:	54
5. DISCUSSION	56
6. CONCLUSIONS.....	57
7. RESOURCES.....	59

INDEX OF TABLES

Table 1 Targeted USB Layers	12
Table 2 Countermeasures.....	34



INDEX OF FIGURES

Figure 1 Attack Modelling.....	18
Figure 2 Targeted Operating Systems	41
Figure 3 Initial Ducky Script Instruction - After Delay.....	42
Figure 4 DuckTape GUI Main Menu	44
Figure 5 DuckTape GUI Captcha Window	45
Figure 6 DuckyScript - VisualAmbiguity.....	50
Figure 7 Attack Result	51
Figure 8 DuckTape Interception	51
Figure 9 DuckyScript Disable Defender and Attack Result	52
Figure 10 UAC - Bypassable (Left) - Secure (Right).....	52
Figure 11 USB Settings - Android.....	53
Figure 12 AutoPlay Settings - Windows 10	53
Figure 13 Interface Overlapping Solution	53
Figure 14 Bypassing Windows Key and Hotkeys to access PowerShell	54
Figure 15 System Benchmark - DuckTape Idling	54

INDEX OF ABBREVIATIONS

Abbreviations:

- BIOS: Basic Input Output System
- CPU: Central Processing Unit
- CVE: Common Vulnerabilities and Exposures
- DoS: Denial of Service
- DDT: Data Delay Time
- GPU: Graphics Processing Unit
- GUI: Graphical User Interface
- HID: Human Interface Devices
- IoT: Internet of Things
- KMCS: Kernel Mode Code Signing
- LED: Light-Emitting Diode
- NIST: National Institute of Standards and Technology
- OS: Operating System
- PID: Product ID
- RAM: Random Access Memory
- TOCTTOU: Time of check to Time of use
- UAC: User Account Control
- UKIP: USB Keystroke Injection Protection
- USB: Universal Serial Bus
- VID: Vendor ID
- VM: Virtual Machine
- WPF: Windows Presentation Foundation

1. INTRODUCTION

1.1 Overview

The way we connect and transfer data between different devices is immensely elevated by the inception and widespread adoption of the Universal Serial Bus ports and protocol as it introduced convenient and efficient method of reliable interaction with technology. However, alongside the benefits it brings, the use of USB also created a bunch of exploitable significant security vulnerabilities. These concerns prompted the development of various security measures which employ different approaches and tools to defend the hosts and end users from potential attacks.

1.2 Problem Statement

The misuse of USB devices and the failure of systems to prevent BadUSB attacks pose significant risks to information security. By granting instant full trust to any connected USB device, systems become vulnerable to potential exploits, including keystroke injections, eavesdropping, and other forms of unauthorized access. The lack of effective defenses against BadUSB attacks and the prevalence of default system settings that allow these attacks to occur underscore the need for robust protection mechanisms.

1.3 Objectives and Research Questions

The primary objective of this thesis is to address the vulnerabilities associated with BadUSB attacks and explore effective countermeasures to mitigate their risks. Specifically, this research aims to:

- Identify the major vulnerabilities underlying the misuse of USB devices and the inherent weaknesses in current system defenses against BadUSB attacks.
- Analyze existing countermeasures and assess their limitations in providing comprehensive protection without compromising usability.
- Design and develop a prevention mechanism, termed "DuckTape," to authorize and secure devices introduced to USB ports, while balancing security and usability.

-Minimizing the threat surface against keystroke injections and enhancing user awareness of potential risks.

To achieve these objectives, the following research questions will be addressed:

What are the key vulnerabilities associated with BadUSB attacks and their impact on information security?

What are the limitations of existing countermeasures and their implications for usability and protection?

How can a prevention mechanism like DuckTape be designed to mitigate the risks posed by BadUSB attacks while maintaining usability and effectiveness?

What is the expected impact of DuckTape?

1.4 Scope and Limitations

This thesis focuses specifically on addressing keystroke injection risks posed by BadUSB attacks and does not encompass the entire spectrum of USB-related vulnerabilities. The research primarily targets Windows 10 operating system environments, considering their wide adoption and prevalence among users. However, the concepts and principles explored may have applicability to other operating systems with appropriate adaptations. It is important to note that while DuckTape provides mitigation tactics against BadUSB attacks, it does not offer a foolproof solution or guarantee complete elimination of the threat.

1.5 Significance

The significance of this research lies in its contribution to enhancing the security of USB-based systems by addressing the vulnerabilities associated with BadUSB attacks. By identifying weaknesses in current defenses and proposing a prevention mechanism like DuckTape, this thesis aims to provide a practical solution that balances security and usability. The research findings and insights gained from this study can benefit end users, security-conscious environments, and the broader field of information security by offering a means to minimize the threat surface and increase user awareness of the risks posed by BadUSB attacks.

1.6 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 explains the fundamental use of USB and identifies the source of the problem and explores BadUSB concepts on different layers highlighting possible threats and key attributes of these attacks, it expands the possible scenarios that expose the lack security on the hosts.

Chapter 3 provides an in-depth review of related literature, discussing the existing research on USB vulnerabilities, BadUSB attacks, and current proposed countermeasures that aim to mitigate the risks, this chapter also identifies the weaknesses of those approaches and locates the main source of vulnerability.

Chapter 4 presents the proposed countermeasure in this research, outlining the design and implementation of DuckTape. It presents the evaluation of DuckTape, including tests, and performance overhead.

Chapter 5 discusses the findings of this research. Finally, Chapter 6 concludes the thesis, summarizing the contributions, discussing future work, and emphasizing the importance of implementing effective countermeasures against BadUSB attacks.

With this thesis, we aim to contribute to the field of information security by offering a practical and efficient solution to mitigate the risks associated with BadUSB attacks. By integrating DuckTape as a prevention mechanism, users can exercise greater caution when dealing with USB devices, safeguarding their systems and enhancing overall security.

2. BACKGROUND

2.1. USB

Prior to studying the attacks presented on the Universal Serial Bus (USB), it is important to understand the basics of USB technology. Computers typically feature an embedded hub, known as the root hub, which have multiple ports. Configurations include a host linked to multiple peripherals that are attached using USB cables (Nissim et al., 2017). The concept of USB eliminated the inconvenience of transferring data between devices. Human Interface Devices (HID) are meant to directly interact with humans and be operated by humans (Charan & Kulkarni, 2022). While standardizing peripheral connections, which can be composite to enable multiple functionalities within one physical USB device incorporating an internal hub, the ease of implementation and availability are direct results of the USB design decisions and success (Mueller et al., 2019).

The USB protocol gained notable adoption and prevalence as a communication method between devices configured to operate in a master-slave style, thanks to its extensive compatibility and Plug-and-Play functionality (Denney et al., 2019). However, this unprotected, vulnerable link lacks any means of filtering, yet it is still perceived as trusted, overlooking the exploits that can possibly occur based on the assumption of granting credibility upon physical proximity. This allows attackers to target the weakness of not implementing any authentication that confirms the legitimacy of a peripheral trying to connect (Gates & Applied Computer Security Associates, 2010), hence by allowing seamless integration through Plug-and-Play to a diverse range of peripherals along with the ease of use on all compatible hosts is both the greatest advantage and the biggest vulnerability of the USB protocol (Tian et al., 2016)

The absence of proper access control methods in the initial phase of enumeration plays a serious role in the weakness of the USB protocol, accompanied by the complexity of establishing reliable means of authentication after enumeration. This results in the host system loading drivers and interfaces without verifying the identity of the device (Tian et al., 2015). Usually certain devices are considered safe for the systems while disregarding how vulnerable the USB protocol is, especially at the enumeration phase (Sikka et al., 2017)

The USB specifications lack emphasis on security measures, as revealed by the Common Vulnerabilities and Exposures (CVE) list (*Common Vulnerabilities and Exposures*, n.d.), which holds over 400 records related to USBs. The priority associated with peripheral security was least considered (Lu et al., 2021). Furthermore, the Human Interface Device (HID) class definition aided in merging different devices onto a universal port, resulting in convenience but leaving protective measures out of the equation with no reliable mechanism to properly identify the specific type of device. The subsequent USB revisions brought no significant value in terms of safety measures, allowing the USB protocol to serve instead as a method for delivering payloads (Lu et al., 2021).

Almost all modern operating systems will accept any self-proclaimed HID keyboard automatically upon detection since peripherals are not complex enough to provide significant authentication. It is rare to have secure ones paired with signed firmware. But to effectively utilize an environment, HID devices require drivers to be installed to ensure their functionality (Charan et al., 2023).

USB devices offer reliability and convenience while using the USB port as their source of power (Mohammadmoradi & Gnawali, 2018). The portability, ease of access, and compatibility of USB devices play a huge factor in allowing them to become beneficial for everyone. However, they also present a dangerous threat that compromises system protections (Yang et al., 2016). The issue of malicious devices designed to compromise their own host is not acknowledged by current systems, and users seem unaware of the extent and danger this problem poses, while these malicious devices are fairly cheap and very capable (Angel et al., 2016).

While all of the traffic remains unencrypted, there is less deterrence from exploiting many of the utilized IoT devices. This also leaves an appealing gap for the interception of data caused by the shortcoming in both authenticating and encrypting the data transmitted over the protocol (Dumitru et al., 2022).

It is arguable that BadUSB is not considered a technical flaw or even a vulnerability since it aligns with the proposed standards of USB, which indirectly allow alteration of peripherals' identities in use while simultaneously keeping the host unaware of the exact number of endpoints being served (Nissim et al., 2017).

Often, the exploitative characteristics of attacks rely on the instinctive trust offered by default habitation in operating systems, which allows for an effective delivery tool to inflict significant damage (Nicho & Sabry, 2022). While users instantly trust the devices to perform according to their apparent intentions, they are in fact simply microcomputers with the competence to execute any programmable action (Nissim et al., 2017) as the USB protocol remains without a significant reliable identification method (Mueller et al., 2019).

Currently, the USB standard exhibits excessive trust within the initial enumeration process, and the lack of enforcement of the implementation of unique serial numbers for peripherals abolishes the ability of the high-level kernel to reliably tell them apart (Kang et al., 2017). Consequently, USB hubs blindly rely on the information provided by the devices, leaving room for possible exposure to a wide range of keypress injection attacks that found their way between benign-looking devices (Neuner et al., 2018). This highlights major security concerns in the software engineering aspect of the USB protocol, as devices are expected to correctly address their true identity and functionality without questioning the integrity of the provided information (Angel et al., 2016).

2.2. BadUSB

The flaws of the USB protocol make it an attractive target (Dumitru et al., 2022). While raising worrisome problems, the most prominent attack, coined "BadUSB" exploits the USB protocol (Mueller et al., 2019). Purposeful and aimed BadUSB attacks form a significant threat to industrial facilities (Zuyeva & Ordabayeva, 2022). USB-based attacks can also pose organization-wide security threats by maximizing the attack surface (Kang et al., 2017). BadUSB is regarded as the most notable IT vulnerability ever identified (Sikka et al., 2017)

Often, USB attacks take advantage of the trust-by-default vulnerability of the USB ecosystem (Nicho & Sabry, 2022), by manipulating the absence of an authenticating process (Dumitru et al., 2022; Lu et al., 2021). Operating systems confidently enumerate these hidden functions and load their drivers, granting them access without further impediment (Tian et al., 2016). The lack of authentication enables connecting devices to conceal their real identities and disguise themselves as other types of USB

devices (Gates & Applied Computer Security Associates, 2010). This absence of authentication and the unrestricted access contribute to the inherent riskiness and threat of USB usage (Yang et al., 2016). Such inexpensive yet powerful devices often exceed the users' awareness of the extent of the problem (Angel et al., 2016) and by taking advantage of the users' lack of awareness of the risks associated with seemingly benign devices (Nissim et al., 2017) USB security became a significant problem as a result of its trust-by-default characteristics (Lu et al., 2021)

Firmware is stored in non-volatile memory that allows for the execution of software instructions (Charan & Kulkarni, 2022) Malicious devices can hide their true functionality through rewritten firmware (Lu et al., 2021) This is due to the lack of integration of firmware signing. BadUSB is a popular category of firmware attacks that alter the device firmware with malicious intent (Lu et al., 2021). This threat hides in the core firmware, it resides at the inner layers of the hardware, which govern the device's functionality (Blanchet, 2018) Firmware attacks modify the firmware stored in USB controllers, allowing for the injection and execution of malicious codes (Fatos et al., 2016). BadUSB is an attack where the firmware of the device is falsified, granting adversaries the ability to carry out malicious actions (Hammoudeh et al., 2021). The firmware branded by Phison can be reprogrammed. Eliminating the firmware by resetting the device or executing a USB reformat is not possible (Blanchet, 2018). Mostly, these peripherals possess more than one internal function, which are referred to as interfaces. Connecting an unfamiliar interface to a host without the user's awareness is the shared characteristic these attacks have in common (Tian et al., 2016). Repeatedly, BadUSB devices are able to perform the registration process multiple times without user permission, given a new temporary address each time from the system (Sikka et al., 2017) as USB's mounting and synchronization capabilities can be abused to run malicious code (Gates & Applied Computer Security Associates, 2010). However, it is challenging to require manufacturers to acquire signed firmware and force the host systems to verify their certificates accordingly because there is a notable number of devices already in the market lacking these specifications. (Mohammadmoradi & Gnawali, 2018)

Commonly mimicking Human Interface Devices (HIDs), plays a significant role in these attacks (Lu et al., 2021). The presence of malicious HIDs introduced significant difficulties (Nicho & Sabry, 2022). Manipulation of HIDs is more common due to their

ability to access the host machine (Nicho & Sabry, 2022). Malicious firmware has the capability to allow a USB device to operate discreetly as a HID (Tian et al., 2015). The HID attack vector stands out due to its unique combination of customized hardware and keyboard emulation (Charan & Kulkarni, 2022).

Typically, there is a noticeable lack of protection against reprogramming the device controller in benign devices (Mueller et al., 2019) such as computer mice, which can contain embedded microcontrollers that are responsible for USB interactions with the host controller, equipped with CPUs and bootloaders (Nissim et al., 2017). Technologically altered ordinary harmless peripherals can possess dangerous attributes (Zuyeva & Ordabayeva, 2022). The modification of these microcontrollers turned basic IoT devices into cyberweapons (Nicho & Sabry, 2022). The popularity and availability of USB firmware chips with reflashing capabilities on the market at cheap prices have greatly contributed to this issue (Neuner et al., 2018) like the Teensy USB development board (Tian et al., 2015). The Rubber Ducky is a notable example which is able to automate a wide range of actions with physical access (Nissim et al., 2017). PHUKD, an abbreviation for Programmable HID USB Keyboard/Mouse Dongle (Blanchet, 2018) shows the modular nature of these devices and allows for the addition of microSD cards where commands can be stored, or Wi-Fi chips to execute scripts remotely (Blanchet, 2018).

The concept created multiple layers of threat vectors and attack paths (Nicho & Sabry, 2022). BadUSB can be classified into four categories: embedded malware, fraudulent descriptors, unauthenticated functions, and modified functions (Takehisa et al., 2017). Resulting types of attacks include obtaining unauthorized access, compromising confidentiality and integrity, on-path attacks such as tapping and logging, off-path attacks such as intercepting the link between other connected devices, electrical attacks, and masquerading. Different attack scenarios include the injection of keystroke commands and file transfer hijacking, as demonstrated in (Dumitru et al., 2022). Hijacking file transfers facilitates the compromise of an OS image during the booting process by a USB storage device. Researchers also classified BadUSB into two categories: driver exploitation and user emulation (Mueller et al., 2019). While addressing other forms of USB-based attacks, which include Denial of Service (DoS) and protocol masquerading (Lu et al., 2021).

The speed of the peripherals is determined by the USB bus speed (Tian et al., 2015). Devices like the Rubber Ducky can emulate predefined keypresses at extremely high speeds, taking advantage of the system's USB port and device drivers' maximum rate (Neuner et al., 2018). Keystroke injections can achieve speeds up to 2000 words per minute (Arora et al., 2021). Usually unnoticed by the users, such attacks happen in brief seconds, and even if noticed, it is probably too late to stop such a threat (Sikka et al., 2017). Recently, as hardware improved, it expanded the capabilities of USB-based attacks beyond previous limitations (Gates & Applied Computer Security Associates, 2010).

Peripherals can act as keyboards, injecting surreptitious keystrokes that carry out malicious actions (Kharraz et al., 2019). BadUSB can willfully emulate human typing patterns, enabling keystroke injection attacks (Denney et al., 2019), while circumventing security protocols and protection policies by posing as an end-user who has maximum control over the system (Arora et al., 2021). USB protocol allows these devices to execute malicious commands, which includes downloading malware or even generating it in real-time (Griscioli et al., 2016). The threat actor can impersonate users while running harmful scripts, installing viruses, and exfiltrating private data from disks (Lu et al., 2021; Angel et al., 2016). Such abusers have employed keystroke and mouse click injection techniques as part of their attack (Nissim et al., 2017).

All actions a typical user performs are possibly automated using BadUSB and executed with identical privileges to the presently logged-in user (Wrightson, 2014). Carrying actions with escalated privileges can be deliberately exploited with malicious intent (Nicho & Sabry, 2022). Studies advise to enforce the least privilege principle on USB devices, especially for device drivers (Tian et al., 2015).

USB's ubiquity makes it an attractive entrance for various attack vectors aiming at organizations (Neuner et al., 2018). An infected host can compromise any connected peripheral device. Resulting devices can serve as instruments for the spread of malware, presenting a compelling security threat (Mueller et al., 2019) and allowing the misuse of the Universal Serial Bus protocol to become even more prevalent (Nicho & Sabry, 2022). On the market, there is an approximate 50% chance that the microcontrollers available are susceptible to this exact flaw, which indicates the need for further assessment. Such a threat goes beyond simple flash drives and can possibly spread to any USB peripheral (Blanchet, 2018). USB peripherals have been exploited for various cyber-attacks for

years now, mainly by leveraging their vulnerabilities, properties, and capabilities (Nissim et al., 2017).

Rogue implants are used for cyber operations on a higher scale of national security. The Stuxnet attack was a red flag and warning signal in regards to the potential use of USB and other embedded devices when employed to compromise any air-gapped networks and systems (Kierznowski & Mayes, 2016). Hardware keyloggers and keyboard-emulating devices are some of the hardware implants that BadUSB can mimic, with the ability to combine such attacks simultaneously (Blanchet, 2018). Challenges arise to prevent unauthorized access and detect it without visual and physical inspection of the USB ports (Kierznowski & Mayes, 2016). BadUSB remains a major weakness that requires effective countermeasures against the exploitable concepts of the USB standard (Charan & Kulkarni, 2022). Exploiting the USB protocol with BadUSB is considered a crucial flaw that has not been addressed properly with enough attention (Charan et al., 2023).

2.3. Multiple Attacks on Multiple Layers

Merely addressing keystroke injection means overlooking the rest of the possible threats associated with BadUSB attacks, it is necessary to acknowledge the ability to run various exploits and resulting in diverse potential fragile points in a system, as these vulnerabilities can overlap (Nicho & Sabry, 2022). The insecure USB bus establishes a direct link that allows attackers to exploit numerous vulnerabilities. This bypasses most security policies and grants privileges imitating those of end users, who have maximum control over the machine (Arora et al., 2021).

The discovery of multimode attack models (Lu et al., 2021), akin to the ones introduced by the rubber ducky (Hak5, n.d.-a), introduces a wide range of possible threats (Blanchet, 2018). Malicious devices can repeatedly reintroduce themselves during the vulnerable enumeration process, obtaining permissions with different addresses (Sikka et al., 2017). Furthermore, the lack of encryption in data transmitted using the USB protocol magnifies the resulting dangers that can impact unprotected systems when multiple forms of attacks are combined (Lu et al., 2021; Dumitru et al., 2022).

Overwhelming vulnerabilities with the universal serial bus protocol suggests that any device which identifies itself as a monitor is allowed to capture video stream from the host (Lu et al., 2021). More threats include Man-In-The-Middle (MITM) attacks, which

include eavesdropping (Kierznowski & Mayes, 2016), modifying, replaying, fabricating, and exfiltrating information sent between hosts and devices over the insecure connection (Dumitru et al., 2022). Currently nothing stops BadUSB from mimicking the role of hardware implants such as keyloggers (Kierznowski & Mayes, 2016) which can come paired with password sniffers (Charan et al., 2023) and credential harvesting (Nicho & Sabry, 2022). Other threats involve network spoofing (Fatos et al., 2016; Blanchet, 2018; Nohl et al., 2014), DNS redirect (Arora et al., 2021), protocol masquerading (Lu et al., 2021), Denial of Service (DoS), changing system settings (Nissim et al., 2017), display spoofing (Nohl et al., 2014), juice filming (Lu et al., 2021) and juice jacking are all serious threats that could be bundled with BadUSB attacks. Moreover, spawning a shell over the network (Kierznowski & Mayes, 2016) which allows for remote execution of scripts (Blanchet, 2018; Cannols & Ghafarian, 2017) and the ability to have full control over the victim device (Lu et al., 2021) by creating backdoors (Charan et al., 2023) or calling home over the network by initiating reverse shells (Neuner et al., 2018; Nissim et al., 2017) are examples of USB attacks.

Files exfiltration as demonstrated in (Dumitru et al., 2022) includes copying out private information from the devices storage (Lu et al., 2021) and stealing valuable data (Yang et al., 2016). Remarkably, data can be exported by encoding it in binary format and transmitting it through the signals associated with keyboard LEDs, such as NumLock, CapsLock, and ScrollLock (Angel et al., 2016) which was also exploited to collect and store data too by tracking keyboard LED states (Clark et al., 2011; Kang et al., 2017).

Furthermore, virtual machine Breakout (Nohl et al., 2014), buffer overflows (Tian et al., 2015; Blanchet, 2018) where the creation of an artificial hub with active ports to exploit a flaw results in authorizing the execution of malicious software, propagation and self-replication of malicious code (Gates & Applied Computer Security Associates, 2010 ; Griscioli et al., 2016). Accidental leaks of electromagnetic signals within the data transmission protocol possibly could contain sensitive information, such as electromagnetic emissions. These emissions are abused with RF transmitters as seen in TURNIPSCHOOL and USBee (Lu et al., 2021). Observations show potential theft of data through non-audible channels using a set of speakers by abusing unintended channels as a pathways (Kang et al., 2017) which probably should not be disregarded, as these threats can occur within a matter of seconds (Nissim et al., 2017). BadUSB can serve as a payload delivery method, coupled with zero-day vulnerabilities and critical

flaws, enabling not only remote delivery of malicious keystrokes and emulation but also full control (Lu et al., 2021).

Table 1 Targeted USB Layers

Vulnerability	Targeted Layers
Keystroke Injection	Data Link Layer
Keystroke Sniffing	Data Link Layer
Buffer Overflows	Data Link Layer
Malicious Firmware Injection	Data Link Layer, Protocol Layer
USB Device Spoofing	Data Link Layer, Protocol Layer
Data Tampering	Data Link Layer, Protocol Layer
DNS Redirect	Protocol Layer
Display Spoofing	Protocol Layer
Virtual Machine Breakout	Protocol Layer
Credential Harvesting	Application Layer
File Exfiltration	Application Layer
Remote Execution of Scripts	Application Layer
Malicious Code Propagation	Application Layer
Power Surge Attacks	Physical Layer
Electromagnetic Signal Leaks	Physical Layer
Juice Jacking	Physical Layer

2.4. Threat Modelling

Targeting any of the available operating systems is valid since almost all of them share the habit of blindly trusting any new USB device introduced to the system. Such attacks do not fall short of targeting anyone. Where it can be a widespread attack that aims at the general public masses with the intent of spreading to as many individuals as possible, it is also an option to specifically focus on a particular target for many reasons, like identity theft or harassment.

2.4.1. Insider Threat

We cannot neglect the abilities and knowledge carried by the threat actor; depending on what information they have about the target victim. There are multiple scenarios that can be abused where BadUSB can pose a significant threat. Upon observation of the current proposed countermeasures, a gap in threat modelling is identified, which leads to shortcomings in the protection approaches. Just because an attack falls outside the scope of a specific countermeasure research process, that does not mean it falls beyond the reach of an attacker at all. As an insider threat could openly allow for direct physical access to the host, introducing the malicious device directly is not far-fetched.

If the adversary is aware of the system components such as the operating system, software used to protect against such attacks, hardware, and common exposed vulnerabilities of the target, the threat actor can intentionally allow corrupt devices to bypass the system defenses, which heavily rely on the user's acceptance or rejection of a device (Neuner et al., 2018).

Some of the countermeasures used tend to keep track of already identified peripherals based on their Product ID (PID), Vendor ID (VID), or even the Serial number, which are not truly unique attributes since a lot of devices share the same VID or PID, it is possible to manipulate that fact into faking such attributes on malicious devices.

If the countermeasure heavily depends on HID coupling, as in, it only intercepts if a second keyboard is introduced, that does not stop the attacker from simply swapping or disabling the primary keyboard if he is aware of such protection measures; that is a valid action that we should expect from on-site threats. Sometimes proper knowledge of

the system isn't a priority especially if you can attack all of the systems expected to be used in any environment.

2.4.2. Outsider Threat

An all-inclusive, multi-platform attack can be carried out using off-the-shelf scripts available on the Hak5 GitHub repository (Hak5Darren, n.d.). That shows a single attack which is compatible with multiple systems, usually targeting the most common ones.

The information passed back by the system to any USB device during the enumeration process is used to identify the operating system, which contains data validating the type of operating system it is interacting with. Additionally, Timing analysis of the USB packets can be effective in identifying the operating system of the host (Tian et al., 2015).

This threat can include attacks against air-gapped systems, where infected USB drives can submit a bridge to overcome such a gap, as shown in the Stuxnet attack (Baezner & Robin, 2017), where USBs were introduced with zero-day worm exploits against the Windows operating system. In (Nissim et al., 2017), it was observed that the firmware of the Logitech G600 has sufficient capacity available to accommodate an entire malware package, which shows the opportunity to bridge the separation.

Once the adversary has physical access to the vulnerable device on site, there are fewer restrictions on the possible harm caused. It can contribute immensely to an attack by allowing the introduction of exploits that would not be easily delivered by mistake in realistic scenarios.

People often mistakenly leave their computers unattended for short periods of time, which is arguably an opportunity for potential attackers to capitalize on (Cannols & Ghafarian, 2017). Even for mobile devices, such as in (Lu et al., 2021), where a victim leaves their phone to charge without locking it, allowing BadUSB-C to actively control it.

The USB protocol can also be exploited to provide access to any connected secure drive, although the screen is locked (Nissim et al., 2017). In boot attacks, the threat extends its reach to the kernel of the devices, allowing the attacker to deactivate some defense mechanisms (Kharraz et al., 2019). There are attacks like the RAM dump (Cold Boot), which takes advantage of the capability to boot from an external device in order to export the contents of the victim's volatile memory to a data partition. Alternatively, as soon as the BadUSB has been introduced to the device, the threat actor can simply walk away and leave it there since eventually it is assumed the user will unlock their computer. Later on, the attack can launch when it is triggered as scripted, possibly not instantly as the victim is logged in, but maybe when sensitive data input or banking-related action is detected by the pre-programmed script, which means the attacker can select the best time to execute the attack (Nissim et al., 2017).

Additionally, hardware tampering involves bypassing suspiciously modified ports, tools, and hubs that are introduced as proxy to counteract specific alteration, or even ports that are physically sealed with a plug.

Linux victims are subject to a sophisticated threat that falsifies 'usbmon' outputs by utilizing rootkits that produce system hooks that eliminate any USB packet that has a matching VID with BadUSB from appearing in usbmon (*Usbmon*, n.d.; Denney et al., 2019).

Another possibility occurs when the attacker aims to deal severe damage to the hardware itself by applying electrical attacks that render the devices useless by dosing multiple electrical power surges to the circuit, causing it to partially or completely fail. (Usbkill, n.d.).

While some attacks do not even require a direct interaction with the target device or the user, by abusing the curious nature and interest of the victim, as seen in situations where USB sticks are found lying outside the environment, and expect the users to introduce these malicious devices themselves to the system out of sheer interest (Neuner et al., 2018).

2.4.3. Attack Attributes

Adversaries take into consideration the duration of time required to achieve their task and optional methods of avoiding detection, which introduce varying methods of dealing with the span of exploitation.

Keystroke injection happens very rapidly, and in cases where it is expected to detect the attack as it is happening by analyzing the keystrokes, it could be a little too late to do anything for malicious scripts that work within very few injections, such as RM-RF commands, which cause arguably irreversible data loss for Unix-like systems and do not prompt for confirmation from the user (Rankin, 2009).

It is explained by researchers how anomalies in keypresses can help identify an unusual use of input devices; therefore, to avoid detection, the attack could handle varying amounts of delay to help further mimic the normal behavior of the common user by employing intervals of pauses in between keypresses, keyholds, and releases. Hak5 coined this as "Jitter" (dallaswinger, n.d.)

Threat actors can capitalize on the inability of the user to visually identify the presence of malicious acts, even if they are in front of them. By making sure that the terminal GUI is operating in the background or in a minimized state that is very difficult to spot, or even having it dragged to the corner of the screen, it limits the ability of the user to identify the malice that is occurring. The physical positioning of certain USB ports falls within the outer bounds of the visual perception of the gullible user, which includes computers where ports are located at the rear and concealed from the default view of sight, which allows attackers to carry out their malicious actions discreetly, evading detection. This does not exclude the use of hidden USB extension hubs.

Depending on the resources, utilities, and knowledge acquired by the threat actor, we can classify the possible abilities and impact depending on the advancement level of the applied attack. Off-the-shelf scripts resemble ones that are prepared by random individuals and are available and accessible to anyone, like the online libraries of executable payloads openly reachable to everyone, which offer a very limited surface of attack and lower levels of success, and could possibly be detected and prevented easily

using simple methods of protection and basic countermeasures or some policy tweaks. Advanced custom scripts are often introduced by specialists who know what they are doing and are attempting to achieve a specific target with enough knowledge to inflict a decent amount of damage to the victim. That includes using technology that avoids detection and bypasses prevention methods while employing other kinds of threats as well. However, state-sponsored threats require extreme levels of ability and expert knowledge to achieve the highest levels of success against specific high-profile victims and air gapped systems; this includes using zero-day exploits, acquiring firmware signing codes from distinguished manufacturers, and abusing all possible resources to accomplish their mission, usually in national cyberwarfare.

These attacks do not stop at using the USB itself to extract information; adversaries tend to abuse the BadUSB to introduce a long-lasting connection to the victim's machine by allowing backdoors into their system, which expands to other types of threats later on.

There exists a possibility for attackers to socially engineer IT personnel and regular users around any countermeasures, which is always an option regardless of the protection in place. Although some approaches can be flawed, we cannot entirely disregard them. Most importantly, avoiding such attacks can be challenging, especially when the attacker is aware of the measures implemented against them.

The combination of features, theoretical scenarios, and attack modelling is visually represented in Figure 1 below.

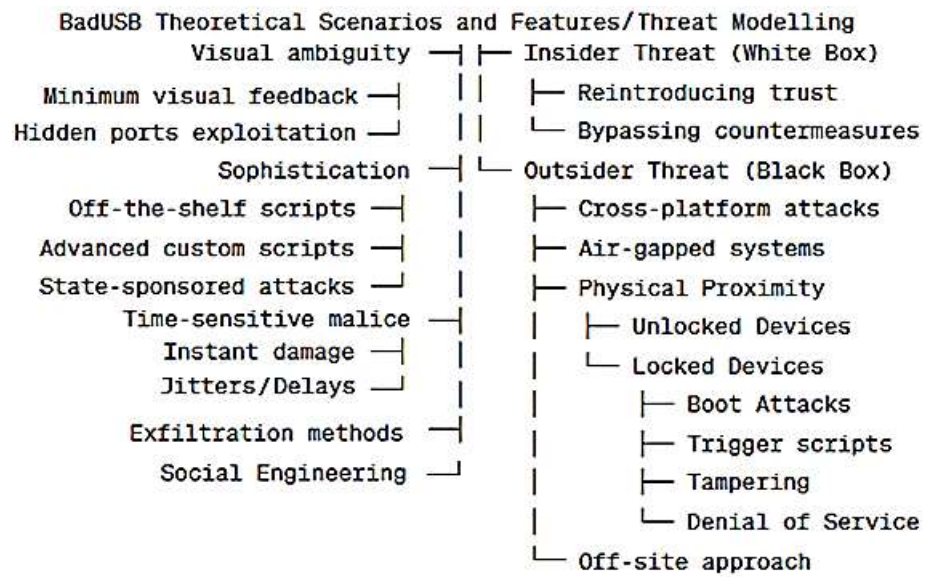


Figure 1 Attack Modelling

3. RELATED WORK

3.2. Software Based Countermeasures Against USB Attacks

GoodUSB

This security mechanism focuses on detecting potential threats during the enumeration phase for the Linux USB Stack. It is designed for both the regular user or a security-conscious businesses that enforce employees to participate in security orientation in order to avoid extreme measures against usage of USB devices. By validating permission to authorize loading drivers limited to the purpose use of the plugged device at the expense of usability as a tradeoff for security, it burdens the end user to identify the expected intent of a plugged peripheral and enforcing least privilege on the device interfaces by disabling undesirable functionalities. As a solution to defend against malicious USB Firmware it tries to detect the mismatch between requested interfaces and the users expectation of the device without performing device fingerprinting thereby claiming that the end user is the most reliable source of information in this situation, it employs a graphical dialogue box that notifies the user of a connection, this interface prompts end users to describe the device, Access to features outside that description is denied, by incorporating a security image system and a virtualized honeypot mechanism which protects the host and observes the device to profile BadUSB attacks and potentially pass it to high level forensic tools for further examination, It controls the additional requests of interfaces from kernel drivers while introducing 5.2% performance overhead to the USB subsystem and 9 milliseconds to the enumeration process as determined by the performance evaluation of the system. It was tested on Ubuntu and Android and claims the defense is sufficiently generalizable and therefore applicable to other systems as well, which expands its potential scope. But by default, this countermeasure assumes that the plugged-in device is benign and is not attempting to evade detection. Hence, this method fails to deliver protection if the threat actor himself is attempting to penetrate the system on site, where it is possible to masquerade as a known device to acquire the desired interface. Another limitation of such an approach is the requirement of a custom kernel (Tian et al., 2015). And it is criticized for being unsuitable as a comprehensive attack detection framework since it lacks cross-platform compatibility for now. Additionally, a keyboard whose firmware was maliciously modified will still be expected to enumerate as a keyboard, rendering

this mechanism ineffective to detect such an attack (Nissim et al., 2017). It's a virtualization-based system that aims to defend against USB threats by offering strong protection capabilities at the cost of altering the user experience by modifying the Linux kernel and using two user space daemons that assess whether a device is considered safe by generating a new virtual machine and inspecting how it reacts to the device. Additionally, the system prompts the end user to make the final decision to accept or reject the new USB device (Mueller et al., 2019). The reported functionality claimed by the device is then presented to the end user. When a device is plugged in, the system blocks its function before enumeration occurs until a series of authorizations are completed (Lu et al., 2021). By utilizing a Linux kernel module that associates devices with designated whitelisted drivers, it involves the user in determining the trust decision between permission and rejection (Neuner et al., 2018). GoodUSB shifts the burden of responsibility onto the user to discern between malicious and benign USB devices (Kharraz et al., 2019). And this solution fails to work in the case that a device employs a vendor-specific interface (Mohammadmoradi & Gnawali, 2018). GoodUSB is susceptible to tampering through rootkits injected into the kernel, which can obfuscate or eliminate evidence of the presence of a malicious USB device, which leads to failure in detection (Denney et al., 2019). Also, it relies on user intervention, but it cannot enforce that the behavior of a device matches what it reports (Tian et al., 2016).

Whitelisting

Attempting to assign a unique, identifiable attribute to each device according to their functionality and characteristics results in a secure list of USB devices that can freely interact and communicate with the host system (Lu et al., 2021). The method is criticized for the heavy reliance (Takehisa et al., 2017) on Vendor ID and Product ID, while some vendors might not apply exclusive identifiers to their devices (Nohl et al., 2014) resulting in ones that are in no way truly unique, making such a solution bypassable and inadequate that also requires the end user to have specific technical abilities and awareness not to mistakenly approve malicious peripherals without caution. Additionally, when introducing a whitelist, it should be considered that hiding the real functionality is an option in BadUSB (Lu et al., 2021). Whitelisting by serial number is not sufficient either since misbehaving firmware is able to falsify its real identity during the enumeration process and present itself as a previously trusted USB device (Tian et al., 2015). Therefore, such general lists are easy to bypass (Zuyeva &

Ordabayeva, 2022). If a threat actor manipulates the malicious device to mimic the identity of a peripheral that was whitelisted for a specific host earlier, the attack will occur undetected. After identifying the challenges proposed with the fingerprinting approach and uncovering the frailty of current whitelisting countermeasures, (Mohammadmoradi & Gnawali, 2018) introduced a more accurate method that considers more identifying features than only VID, PID, and serial numbers, which are not unique enough since multiple devices can share equal attributes that result in them not being distinguishable, which allows for attacks to execute in some cases. More than half of a subset of devices tested had no serial number at all and used base class code, which is easy to spoof. To further enhance the efficiency of this approach, the researcher identified seven attributes of USB devices that resulted in 98.5% success in whitelisting accuracy to validate its effectiveness. Though it is addressed, there is no assurance in the detection of reprogrammed devices in case the attacker was able to locate a trusted device and modify the firmware without altering its interfaces. This makes the approach less reliable.

Code Signing

As mentioned earlier, by default, firmware cannot be scanned (Tian et al., 2015) by the host or other applications (Sikka et al., 2017). When it is possible, it requires the help of the firmware itself, which can be spoofed in the case of malicious firmware (Nohl et al., 2014). The lack of protection that ensures firmware upgrades contain a manufacturer-unique digital signature (Blanchet, 2018), which matches with the reinforced signature database (Angel et al., 2016), as seen in Kernel Mode Code Signing (KMCS), which requires authenticode certification (Pham et al., 2010). It is also observed in (Yang et al., 2016) that digital signature is a technique suitable for specific environments, but it is aided by another security approaches to perform as expected. Yet it is relatively rare to find devices with signed firmware (Tian et al., 2015) because manufacturers do not integrate it onto their hardware, expecting the users to identify and trust the devices (Nissim et al., 2017), which indicates the low probability of the wide spread of this protection method caused by current industrial practices (Blanchet, 2018). Similarly, IronKey (Gillespie, n.d.) was criticized for being costly and difficult to integrate in a lot of environments as the keys could be either shared or stolen (Nissim et al., 2017). IronKey could assist in the prevention of BadUSB attacks but it heavily relies on manufacturers' trust and ability to safely keep certificates and keys protected. Unable to

protect an enterprise environment against state-sponsored attacks where the manufacturers were obligated to share vital signing keys that led to the success of the Stuxnet attack (Tian et al., 2015), which raised doubts regarding the integrity of this approach.

Antivirus

Current antivirus software is not positioned for detection or defense against such attacks (Tian et al., 2015; Fatos et al., 2016). Antivirus lacks the ability to access the firmware running on the device, where BadUSB hides its malicious functionality (Blanchet, 2018; Zuyeva & Ordabayeva, 2022; Cannols & Ghafarian, 2017). Such an abstraction layer makes antivirus unable to cope with this attack (Neuner et al., 2018), making it an ineffective method of protection in this context (Griscioli et al., 2016). Antivirus software cannot distinguish BadUSB from ordinary devices (Lu et al., 2021). It can only be helpful in detecting malware loaded on devices. Commercial software vendors will show a popup on every device the user attaches (Mueller et al., 2019). Like Kaspersky Endpoint Security 10 for Windows which appears to have a feature that attempts to prevent BadUSB attacks (KasperSky, n.d.), the option works by locking the newly introduced keyboard until authorized, but it is worth noting that Kaspersky was banned in multiple countries and is a paid monthly subscription of \$30 without introducing unique methods of detection.

Heuristics

It is also important to acknowledge the ability of attackers to mimic user typing patterns (Mohammadmoradi & Gnawali, 2018), rendering detection based on behavioral patterns difficult (Zuyeva & Ordabayeva, 2022). Yet analyzing the duration of keypresses and speed of typing (Denney et al., 2019) could be helpful to detect injection attacks. (Arora et al., 2021) Introduced The Ducky-detector, Designed to detect and protect against malicious devices with no false positives and required minimal computational power, it offers high accuracy while imposing only a negligible 0.9% overhead on a Linux system. By converting enumerated data into dictionary items, enabling further analysis by the heuristic checker. When two or more devices are enumerated, a user confirmation is generated to verify the credibility of the registered devices. If a registered device's keyboard type, such as the number of function keys, does not match the actual keyboard, an alert is raised. Upon passing both of these checks, the software

analyzes a number of keystrokes every 10 seconds. By capturing raw keystroke data and segregating it into keypress speed, key release, and key hold durations, the software provides a comprehensive analysis of keyboard activity. In the event that any of the checks fail, a warning message is raised to promptly alert the user of a potentially malicious device. The software ensures privacy by shredding any logged data, mitigating privacy concerns. Additionally, it can be easily integrated into existing security infrastructure without significant modifications. However, it's important to note that some antivirus software may mistakenly flag the Ducky-detector as a virus due to its logging and data enumeration capabilities. Nonetheless, the Ducky detector outperforms most existing solutions as a platform- and hardware-independent detection tool (Arora et al., 2021). While it shows alerts to notify users, it does not actively stop an ongoing attack.

USBBeSafe

To identify BadUSB-style attacks, USBBeSafe relies on analyzing how benign devices interact with the host and the operating system. By leveraging insights from millions of USB Request Blocks, the researchers carefully analyzed the labeled data and narrowed it down to three feature categories: content-based, timing-based, and type-based features. They trained different machine learning techniques, finding the most accurate algorithm for building the detection model. This makes anomaly detection a promising direction to defend against BadUSB. The researchers tested USBBeSafe against multiple adversarial scenarios to test its resilience to evasion. With the assumption that adversaries have significant freedom to generate new forms of BadUSB-style attacks to evade detection, they found that USBBeSafe can successfully detect mimicry attacks with different levels of sophistication. USBBeSafe is a lightweight operating system service that can be implemented as either an early warning solution notifying admins or to identify BadUSB attacks and disable the offending port. The architecture of a USBBeSafe-enhanced system requires interactions among multiple components of the operating system. When a malicious device is detected, USBBeSafe notifies the user as well as kernel space components to block the corresponding interface. The service requires two to four training days to keep the true positive rate above 93%. Researchers cannot provide any strong security guarantees to protect multi-user machines or produce a low false-positive rate as they do not have enough data to make any statistically significant claim on accuracy. But as an anomaly-based detection system that relies on

machine learning, where the detection results depend on the quality and volume of the trained dataset, also, an attacker can try to imitate benign USB traffic patterns and evade the detection mechanism. Additionally, an attacker could automatically inject artificial delays to decrease the novelty score of USB traffic. Finally, USBBeSafe fails to protect against scenarios where an adversary attempts to trick users into voluntarily disabling the service by mimicking the output of USBBeSafe and forcing the user to disable protection. The researchers noted that these issues are fundamental to any host-based protection tool. (Kharraz et al., 2019) USBBeSafe performs analysis on the data collected from USB request blocks; according to machine learning algorithms, it detects malicious packets and alerts the user (Lu et al., 2021). Inspecting the performance and behavior of the device and examining potential threats by employing machine learning may still fail to detect attacks where intentional imitation of human typing dynamics is applied (Denney et al., 2019).

Context and Provenance

The aim of this approach is to enhance security without altering the user experience. Which focuses on regular users' systems rather than high-security environments. The proposed strategy involves a non-intrusive baseline defense mechanism against USB attacks, aiming to defend without involving the user. The assumption is made that the user cannot connect a device to their machine when they are not present. The approach includes considering the provenance of dangerous keystrokes to deflect traditional BadUSB attacks and making automated security decisions. To prevent unauthorized access, external keyboards are not allowed to press keys such as Ctrl, Alt, or Super, which can be used to spawn a terminal. If a second keyboard attempts to press these keys, the user is informed about the blocked keys and provided information on how to escalate the keyboard's privileges. The legitimacy of context and provenance rules is based on real-life usage data from USB devices. By following security-by-design patterns, this method minimizes the need for user interaction. It can be directly integrated into the USB stack of all operating systems at the protocol layer, which governs host-device communication. It should be noted that the data used for analysis was collected from a potentially unreliable source (Pastebin) and may be biased toward malfunctioning systems. While the sample size is small and prevents making definitive claims, the findings serve as strong indicators. The proposed defense mechanism is released under a free software license and has been proposed for inclusion in a major

desktop system. However, further formal evaluation is necessary to ascertain if the goal of protecting users without their knowledge can be achieved (Mueller et al., 2019).

By handling new input devices naïvely while filtering keystrokes originated from newly attached devices and either blocking the device if there is another keyboard already plugged, or if it is not a keyboard. or locking the screen (Lu et al., 2021) , Yet it might not include protection when the user is using computer.

USBLock

An implementation that aims to enhance system security by detecting suspicious USB devices through the analysis of temporal characteristics in their generated USB packet traffic. The system is designed to operate without requiring user involvement in the trust decision process, thus reducing the dependency on user actions for maintaining system security. The proof-of-concept implementation of USBLock is specifically developed for Linux systems. Its primary goal is to simplify attack detection and offer immediate neutralization upon device connection by achieving fast detection and removing the user from the security decision loop. To monitor USB connections and collect relevant USB packet traces, the researcher utilized the Wireshark network protocol analyzer. USBLock distinguishes between normal typing behavior exhibited by human users and abnormal keypress sequences. In the event of detection, the system performs actions such as switching off the power to the port and unloading the device driver, occasionally interrupting the flow. It is important to note that USBLock is resistant to attacks involving malicious firmware that attempt to delay payload launch. However, it does not provide defense against attacks launched at the BIOS level. USBLock also includes support for whitelisted YubiKeys, ensuring their proper functioning within the system. In case of detection, it is recommended for IT support personnel to inspect the offending system (Neuner et al., 2018).

By applying a heuristic to detect abnormal keypress sequences and filtering by dynamics, USBLock could block benign devices that are meant to type fast, as was the case with YubiKey, which got whitelisted. This does not scale to unknown future devices with similar characteristics (Mueller et al., 2019) and this included the reintroduction of trust by abusing whitelisting mechanism flaws. Such an approach analyzes the potential threat as it is occurring by monitoring the behavior and classifying it, which tends to fail to detect attacks where the malicious device

intentionally imitates the common human typing dynamics (Denney et al., 2019), but it can be argued that an attacker will attempt to inject keystrokes in a very short time to reduce the risk of being discovered, rendering such an attack detectable (Lu et al., 2021). Also, it is one of the protection measures that can be tampered with through rootkits in the kernel, which hide the evidence of existing rogue USB devices (Denney et al., 2019).

USBFilter

Introduces the first packet-level access control mechanism for USB, enabling the tracing of individual USB packets to their respective processes and facilitating the prevention of unauthorized access. Unlike previous works, USBFilter offers a higher granularity for packet filtering, allowing for enhanced security measures. The latency increase introduced by USBFilter is minimal, ranging from 3 to 10 microseconds. With its packet-level firewall capabilities, USBFilter provides tracing of packets back to the source or destination process and interface, contributing to preventing unauthorized access and data exfiltration. However, it is worth noting that some rules of USBFilter rely on Product ID (PID) information, which may not always be reliable. USBFilter can potentially be vulnerable to time-of-check-to-time-of-use (TOCTTOU) attacks. In terms of usability, writing USBFILTER rules requires a certain level of knowledge about the USB protocol. Due to its characteristics and capabilities, USBFilter is better suited for adoption within operating system kernels. Overall, USBFilter offers significant advancements in USB access control and packet filtering with minimal latency impact, although attention should be given to the reliability of PID information and usability aspects (Tian et al., 2016).

Packet-level access control by specific rules that allow or deny functionality at the low level of the operating system includes limitations such as the possibility of an adversary evading the rules by modifying the attack technique, which mismatches the detection rules; therefore, it only limits known attacks while being vulnerable to sniffing and kernel-level malware (Nissim et al., 2017). This system inspects packets and matches policies against the observed traffic, such as preventing overwriting data on an external drive (Mueller et al., 2019). Such a solution requires a major change in the current USB protocol to introduce an access control mechanism that prevents unauthenticated interfaces from connecting to the host by tracing individual packets and limiting or blocking their access (Kharraz et al., 2019). By defining access rules in USBTables

from the user in the Linux kernel, it is still possible for advanced threat actors to evade detection by adjusting their behavior according to the rules since this method does not include anomaly detection capabilities (Neuner et al., 2018).

G-Data Keyboard Guard

Relies on PID, VID (Mohammadmoradi & Gnawali, 2018) to control the access of new keyboards to a system in order to counter any keystroke emulation attacks. (Blanchet, 2018). A newly inserted device is confirmed by the user before it is initialized; it monitors and traps keyboard insert events; if it detects an insertion of an HID keyboard, a popup notification allows the user to decide whether to allow or block the keyboard; when blocked, it does not get initialized; this method is limited to HID keyboards on Windows only, therefore not offering full protection against devices with malicious firmware emulating anything other than a keyboard (Kang et al., 2017). G Data Guard software relies on the user's decision by asking the user to trust or not, followed by whitelisting according to the decision made, which allows reintroducing trust if a threat actor implements the attributes of a previously whitelisted identifier to inherit authorization (Neuner et al., 2018).

Qubes

Is a Linux distribution that protects the OS and applications from malicious USB devices at the expense of functionality (Angel et al., 2016), a system that attempts to defend against USB-borne threats by virtualization-based solutions and offers strong protection capabilities at the expense of a changed user experience (Mueller et al., 2019).

Secondary Device Detection

This method is explored in (Kierznowski & Mayes, 2016) and implemented in (Arora et al., 2021) and (Mueller et al., 2019). When more than one keyboard in total is enumerated to the host, it is considered a threat and alerts or directs the user to authorize or block the new HID. This approach can be argued against if the threat actor swaps the devices on purpose by unplugging the trusted keyboard to carry out their attack. It will fly under the radar if this is the only approach applied.

Cinch

This is a virtualization based approach that impacts the user experience (Mueller et al., 2019)., With similar principles to USBFilter, it isolates all devices from the host and allows communication through a virtual machine as a gateway to enforce access policies (Neuner et al., 2018). It leverages a unique approach by introducing as little change as possible to the user's operational status (Kharraz et al., 2019). This pragmatic response utilizes virtualization to attach devices to a logically separate and untrusted machine, seamlessly integrating existing operating systems with low overhead. The approach employed by Cinch aims to make USB devices appear remote by preventing direct interaction with the computer while also introducing a choke point for defence without modifying existing standards. This portability across systems, coupled with its flexibility and low overhead, makes Cinch an appealing solution. With hardware-assisted I/O virtualization, Cinch adds less than 3 milliseconds of latency. It enforces logical separation at the boundary between the host controller and its driver, ensuring robust isolation. Furthermore, Cinch enhances its capabilities by incorporating a signature database, leveraging cryptography, and enabling logging. However, it should be noted that Cinch may face limitations in dealing with audio and video devices through cryptography between environments, requiring certificates or modifications to the host devices for proper pairing (Angel et al., 2016).

FirmUSB

FirmUSB has a slightly different focus, namely to analyze firmware currently running on a USB device and determine whether it is benign. It is a framework for firmware analysis that uses symbolic search techniques to examine firmware images (Kharraz et al., 2019). Unfortunately, the fact that their software is not freely distributable prevents widespread adoption (Mueller et al., 2019).

DuckHunt

DuckHunt addresses the risks associated with simple keystroke injection techniques. Its main goal is to mitigate these risks by monitoring and analyzing the timing of keystrokes received by a computer system, whether from a legitimate keyboard or a device posing as one. Through the analysis of time intervals between consecutive keystrokes, DuckHunt is capable of identifying instances where these intervals are unusually short, indicating a potential attack. DuckHunt provides users with a variety of defensive measures to counteract such threats. These measures include options to

disable the keyboard with a password, temporarily interrupt the keystroke sequence, disrupt it, or simply log the event for future reference. However, it is important to note that while DuckHunt effectively addresses simple keystroke injection attacks, it may not be as effective against more advanced tools that employ longer delays between keystrokes or can simulate human keystroke dynamics. The detection algorithm employed by DuckHunt follows a series of steps. Initially, it intercepts keyboard events to capture incoming keystrokes, assigning each keystroke a timestamp for future analysis. When a subsequent keystroke is detected, DuckHunt calculates the Delta Delay Time (DDT) between the previous and new keystrokes, storing the computed value in an array. The algorithm then examines this array of DDTs, specifically checking for a predefined condition. It should be noted that users have the flexibility to adjust the default values of consecutive DDTs and the threshold, respectively, according to their specific needs and requirements. (Farhi et al., 2019; pmsosa, n.d.)

UKIP (USB Keystroke Injection Protection)

Google has proposed a tool that can help prevent keystroke injection attacks. UKIP works by measuring the timing of incoming keystrokes and determining if they are an attack based on predefined heuristics. The tool offers two different modes of operation: monitor and hardening. In monitor mode, UKIP will not block a device that is classified as malicious but will write a log line with information about the device to syslog. In HARDENING mode, UKIP will immediately block a device that is classified as malicious or attacking. A challenge when running UKIP is the proper selection of the two main heuristic variables, keystroke window and abnormal timing, which control the behavior of the tool and its detection capabilities. The keystroke window is the number of keystrokes it looks at to determine whether it's dealing with an attack or not. Based on internal observations, 5 is an effective value for the keystroke window, and 100 milliseconds is an effective value for abnormal timing. However, these values may need to be adjusted based on the specific user's experience and typing behavior. UKIP is available on Linux systems and can be found on GitHub. (Google, n.d.-a, n.d.-b)

3.1. Hardware Based Solutions

TMSUI

TMSUI is a Trust Management Scheme for USBs targeting Industrial Control Systems. The proposed scheme focuses on securing small enterprise environments by accurately controlling USB devices based on security chips and digital signature techniques. The scheme achieves this by customizing an offline whitelist for authorized USB storage devices, enabling them to access protected terminals. The whitelist, which is protected by a security chip embedded within the protected object, is verified for authenticity. Taking into consideration various application scenarios, the TMSUI security chip offers flexibility in authorizing devices to access specific terminals within an industrial control system for a designated period. Through evaluation results, it has been determined that it successfully meets the security goals with high compatibility and good efficiency. Administrators can monitor and allow only trusted devices equipped with security chips, ensuring enhanced security within the environment. Although it is claimed to be difficult to bypass, the system is not protected against cracking the security chip or breaking cryptographic algorithms. also introduces an overwhelming load on centralized administrators, who can possibly be corrupt personnel who have complete control over this system (Yang et al., 2016). Even though TMSUI protects ICS by limiting terminals and their duration of use (Neuner et al., 2018), depending on the rich knowledge of administrators in constructing a whitelist, modified devices can hide their real functionality (Lu et al., 2021), and the system fails to prevent those malicious devices from requesting additional interfaces that introduce potential attacks (Nissim et al., 2017).

USBCheckIn

Another hardware-based implementation designed to enhance security by requiring physical user interaction with the USB device before allowing its use. Which can be utilized with any host, including during the boot process. To ensure usability, the system provides instructions to the user through messages displayed on its own interface. The design of the user interaction aims to maintain high usability while minimizing the possibility of brute force attacks. USBCheckIn does not rely on a malicious device for authentication and is compatible with various BIOS, boot loaders, operating systems, and hosts. It has the capability to simulate a physical disconnection

from the host by sending a detach signal, which triggers a new enumeration and presents different class and type information from the previous enumeration. The CPU consumption of USBCheckIn is negligible, and it offers strong protection against brute-force attacks. However, USBCheckIn has limitations. It cannot prevent a malicious Human Interface Device (HID) from allowing the user to enter the challenge code, and there is no mechanism in place to prevent an attacker from entering the code (Griscioli et al., 2016). Which renders this approach somewhat easy to bypass.

Being dedicated hardware that sits between the host and the USB device, authorization is conducted on the external hardware (Lu et al., 2021), which truly minimizes the use cases where this method can be most usable compared to other approaches.

USBWatch

This solution enhances USB device monitoring by utilizing dedicated, independent internal hardware placed between a USB device and the host machine. This hardware enables the capture of raw, unaltered USB protocol communications. The captured data is then fed into a machine learning-based classifier that dynamically determines the true nature of the USB device based on various factors such as the duration of key presses and typing speed. What sets USBWatch apart is its ability to capture unaltered device behavior while consuming fewer computing resources compared to software-based approaches. In terms of the threat model, the researchers assume the presence of an attacker with physical access to a computer but limited time to initiate an attack, such as a rogue employee at a secure facility. It is assumed that the attacker cannot tamper with the computer and remove the USBWatch hardware since doing so would require opening the computer and breaking the embedded circuitry (Denney et al., 2019).

For device creators and manufacturers, this may be a viable option to consider since the board used by the researcher to test USBWatch costs less than \$20, which might result in a minimal increase in price per unit. However, it is important to note that such solutions might not be easily affordable for the security managers of establishments. Since these solutions leave older hardware vulnerable, they fail to provide a comprehensive and effective security solution. Consequently, a large number of devices are excluded from this protection measure.

USBCondom

A hardware solution that enhances security by blocking data channels through the addition of a blocker in the USB connector, this blocker specifically cuts off the connection of the D+ and D- lines while maintaining the integrity of the power pins. However, the implementation of this method presents a significant challenge to the plug-and-play functionality of USB. Once deployed, the USBCondom restricts all USB functions except charging, which hampers the device's ability to perform its intended operations. While this approach achieves a high level of security, it distrusts all types of USB devices by blocking data channels indiscriminately. Despite its effectiveness in preventing USB attacks, it is crucial to note that threat actors can bypass this defence mechanism relatively easily. They can exploit the other exposed ports or simply remove the USB Condom hardware to gain unauthorized access to the system (Lu et al., 2021).

USBWall

Arguably comparable to commercial anti-virus software (Cannols & Ghafarian, 2017), USBWall uses a beagle board to enumerate devices on behalf of the host and involves the user in the trust decision (Neuner et al., 2018). USBWall utilizes a low-cost, open-source computer called the BeagleBoard to function as middleware, serving as a protective layer between the host and USB devices. This additional middleware enables USBWall to enumerate devices on behalf of the host, effectively sandboxing the USB enumeration process, which often serves as the root cause of vulnerabilities. To create a complete USB sandbox environment, USBProxy is employed as a software platform. The study thoroughly examines various layers to determine the most suitable location for displacing trust. Despite the inconvenience of delayed initialization, USBWall achieves substantial security benefits. By delegating the USB enumeration process to the BeagleBoard, it successfully safeguards the host by providing effective separation from the suspected devices. However, it remains questionable whether USBWall can gain a wider user base due to the minimal changes required on operating systems (Kang et al., 2017). Although it fails to offer full protection if any exposed port is exploited without first testing the device on the board.

According to the table below, It is observed that most Static mechanisms employ some level of whitelisting or PID, VID reliance, and most Dynamic ones employ heuristics to detect threat. And both approaches are no longer strong anymore with the appearance of attacks which use VID, PID swapping and Jitters and delays that mimic human typing attributes. (Hak5, n.d.-b)



Table 2: Countermeasures

Countermeasure	Overview	Features	Drawbacks
GoodUSB - Static	Detects potential threats during the enumeration phase. Validates permission and authorizes loading drivers. Matches user expectation with device driver requests and limits the interfaces	Blocks functions until authorization is complete, Virtualized honeypot mechanism, security image system, Generalizable to other systems, tested on Ubuntu and android	Assumes plugged device is benign. Requires a custom kernel. Heavily relies on end users, Ineffective against masquerading attacks, tradeoff between usability and security, will not stop a keyboard from making an attack, employs a whitelist
Whitelisting - Static	Attempts to assign unique identifiable attributes to each device. Generates a secure list of USB devices by fingerprinting.	exploring more unique attributes other than serial number, if applied it consumes minimal resources	Reliance on Vendor ID and Product ID. Easy to bypass if a malicious device mimics a whitelisted device. Technical knowledge and user awareness are essential
Code Signing - Static	Uses digital signatures to verify firmware authenticity. Requires firmware upgrades to contain unique digital signatures.	verify authenticity and integrity	Firmware not easily scanned. Lack of widespread integration by manufacturers. Relies on manufacturers' trust and key protection, key can be lost or stolen. Can be costly as seen in IronKey
Antivirus	Can detect malware loaded on devices.	Can identify propagating malware, scan and stop the spread	Cannot access firmware, Unable to detect or defend against BadUSB attacks. Therefore ineffective.
Ducky Detector - Dynamic	Analyzes keystroke duration and typing speed to detect injection attacks. Offers high accuracy.	Provides accurate detection of malicious devices. Shredding logs to preserve privacy, checks for multiple keyboards and matches the claimed attributes	Difficult to detect behavioral mimicry attacks. Does not actively stop ongoing attacks. Only raises alerts
TMSUI - Static	Controls USB devices based on security chips and digital signatures. Offers protection for industrial control systems	Provides control over USB devices. And limits access to protected terminals for specific amount of time	Vulnerable to security chip cracking and cryptographic attacks. Overwhelming load and a lot of trust on administrators. Also employs whitelisting.

USBBeSafe - Dynamic	Analyzes benign device interactions and uses machine learning for anomaly detection.	Lightweight and can be implemented as an early warning solution, blocks interfaces when anomalies are detected	Relies on quality and volume of training data. Vulnerable to mimicry and evasion attacks. Requires days of training to achieve decent true positive scores
Context and Provenance - Static	Considers provenance of dangerous keystrokes. Prevents unauthorized access. Minimizes user interaction.	two protection levels : always block (devices that are not keyboard, and if not the only keyboard) and lock the session -by default / if locked OR if the screen is unlocked, authorize	Relies on user presence. Based on data from not reliable sources, possibly no protection if screen is unlocked
USBCheckIn - Static	Requires physical interaction with the device for authentication. Provides instructions to the user through its interface. Employs middleware	compatibility with different systems, works for locked systems and during boot	no mechanism to prevent an attacker from entering the code, potentially easy to bypass, not very practical
Countermeasure	Overview	Features	Drawbacks
USBLock - Dynamic	Analyzes temporal characteristics of USB packet traffic. Switch off power, unload drivers on detection of anomalies	Enhances system security without user involvement. Linux proof of concept	Requires further evaluation. Interrupts flow of user on false positives, employed whitelisting for YubiKeys
USBWatch - Dynamic	internal hardware and machine learning-based classification. Captures unaltered USB protocol communications.	consumes less resources, embedded in the host	Limited affordability for security managers, excludes legacy hardware
USBFilter - Dynamic	inspects packets and matches policies against the observed traffic	Packet-level firewall capabilities, minimal latency increase.	Relies on PID information, potential vulnerability to TOCTTOU attacks, requires USB protocol knowledge
G-Data Keyboard Guard - Static	requiring user confirmation on new keyboards. Monitors and traps keyboard insert events. Tested on Windows.	does not allow initialization of new keyboards until user authenticates it	Limited to HID keyboards, does not protect against devices with malicious firmware emulating other types of devices.
Qubes - Dynamic	Linux distribution , employs virtualization-based protection against BadUSB	Strong protection, as a virtualization based solution.	Functionality impact, changed user experience.
USBCondom - Static	blocks data channels through the USB connector. Maintains power pin integrity. Restricts all USB	High level of security, blocks data channels.	Impacts Plug and Play and all USB functionality, distrusts all USB devices, can be bypassed.

	functions except charging.		
Secondary Device Detection - Static	Considers the presence of more than one keyboard as a potential threat and alerts the user.	Raises alerts for multiple keyboards, user involvement in decision-making.	Can be bypassed if the threat actor intentionally swaps keyboards.
Cinch - Dynamic	Virtualization-based approach that enforces logical separation at the boundary between the host controller and its driver and employ access policies	Isolates devices, minimal user experience changes, low latency impact.	Limitations in encryption dealing with audio and video devices, requirements for certificates or modifications.
USBWall - Static	employs BeagleBoard to enumerate USB devices and sandbox the enumeration process.	effective separation from suspicious devices.	limited protection if other exposed ports are exploited or a device was not first tested on BeagleBoard makes it less practical unless it is implemented on every port
FirmUSB - Static	Firmware analysis framework for examining firmware running on USB devices.	Allows analysis of USB device firmware for potential malicious behavior.	Requires expertise in firmware analysis, limited to devices with accessible firmware, time-consuming process, may not detect sophisticated firmware-level attacks, may not prevent the initial interaction with a malicious USB device
DuckHunt - Dynamic	analyzing of keystrokes received by a computer system. It identifies instances where the intervals between consecutive keystrokes are abnormally short	Provides options to disable the keyboard, interrupt or disrupt the keystroke sequence, or log the event	can be bypassed using longer delays between keystrokes or simulate human keystroke dynamics. May generate false positives or false negatives depending on the settings.
UKIP - Dynamic	measures the timing of incoming keystrokes and determines if it is an attack based on predefined heuristics with the ability to log or block malicious devices.	Provides flexibility in fine-tuning the tool's behavior to control the false positives ratio	limited to protection on Linux systems currently

3.3. Drawbacks

3.3.1. Observed Weaknesses

Several issues have been identified in the currently proposed approaches for addressing security vulnerabilities in the context of device identification and trust. One such weakness is the reliance on the VID, PID, and Serial number (Mohammadmoradi & Gnawali, 2018), which involves trusting the claimed identity (Tian et al., 2016) of a device without considering its ongoing behavior. This approach assumes that devices will always be honest about their actual intentions (Kang et al., 2017; Angel et al., 2016), which may not be the case in real-world scenarios (Tian et al., 2015; Dumitru et al., 2022; Takehisa et al., 2017). Another weakness lies in the concept of trust after authorizing a device once. This approach assumes that once a device has been authorized, it can be trusted indefinitely. However, this overlooks the possibility of devices being compromised, behaving maliciously after the initial authorization, or having malicious devices imitate their attributes.

The use of code signing as a security measure also has its limitations. Codes can be stolen (Charan & Kulkarni, 2022) or forcibly shared with government authorities, compromising the integrity of the system. Additionally, the implementation of cryptography for securing devices can degrade performance, impacting the overall efficiency of the system (Tian et al., 2015; Dumitru et al., 2022; Nohl et al., 2014; Angel et al., 2016). Cost-related factors (Lu et al., 2021), such as licensing (Mueller et al., 2019), technical configurations, maintenance requirements, and corporate policies (Pham et al., 2010), present additional weaknesses. These factors make the application of certain security measures difficult, leading to limited adoption. Furthermore, static detection methods that occur before attack execution (Denney et al., 2019) are insufficient (Lu et al., 2021), as the system remains vulnerable to authenticated devices. Privacy concerns (Arora et al., 2021) arise with heuristic approaches that rely on extensive logging.

Another weakness lies in the lack of widespread acceptance of proposed countermeasures by major operating systems (Mueller et al., 2019; Kang et al., 2017). Despite the existence of various security solutions, many have not yet been adopted by mainstream platforms. This limits their effectiveness and hinders the development of a

unified and comprehensive approach. Additionally, the presence of separate layers of vulnerabilities (Mueller et al., 2019) makes it challenging to address all of them through a single approach. Also, relying on detecting multiple keyboards as an alert for potential attacks can be easily bypassed by removing the primary keyboard before launching an attack. Some proposed approaches are purely theoretical and have not been implemented in practice (Charan & Kulkarni, 2022), leading to adoption difficulties. Furthermore, disabling autorun functionality may compromise plug-and-play capabilities (Karystinos et al., 2019), impacting the user experience. Moreover, presenting challenge codes that can be solved by both users and attackers leaves room for onsite attacks to occur undetected. (Griscioli et al., 2016)

The incidence of false positives, where legitimate actions are incorrectly flagged as malicious, can lead to unnecessary disruptions and inconveniences for users, undermining the trust and effectiveness of the security measures. Both software and hardware approaches have their own limitations. Software-based solutions often consume system resources, leading to CPU overhead and potential conflicts with other applications. They may also be susceptible to rootkit attacks and system hooks on Linux, altering the USB monitoring process (Denney et al., 2019). On the other hand, hardware-based approaches can be costly, tampered with, or removed, and they may exclude older legacy devices (Mohammadmoradi & Gnawali, 2018) that lack the necessary security features.

The "access all or nothing" approach often employed by security measures (Angel et al., 2016) means that once a device gains access or trust, it is granted full privileges without considering the varying levels of risk associated with different actions or functionalities. This binary approach can leave the system vulnerable to attacks that exploit privileged access to execute malicious activities (Tian et al., 2016).

Possibly, if UScramBLE had been introduced at the inception of the USB protocol, it would have been more effective and guaranteed a more protection-oriented space for universal use of devices. Sadly, with the constant improvements to the devices and ports, only the threats grew larger, and none of the proposed protection methods were widely implemented in the main systems to provide comprehensive protection in today's threat landscape (Neuner et al., 2018).

A more feasible strategy would be to increase awareness of possible threats and encourage users to take security-conscious activities and follow the best security practices rather than depending on minor system changes or safeguarding users without their knowledge (Mueller et al., 2019; Kharraz et al., 2019). This collaborative effort between users and technology can create a safer environment. Some approaches directly speak to enterprises and completely overlook the common user with limited technical abilities. Lastly, it is important to note that the proposed approaches primarily focus on a specific attack while potentially overlooking other attack vectors. Additionally, some methods mistakenly block the functionalities of certain devices, such as YubiKeys (Neuner et al., 2018; Mueller et al., 2019), due to misinterpretation or a lack of consideration. In light of these weaknesses, it is crucial for new approaches to address these shortcomings and strive for improved effectiveness, usability, and compatibility with existing systems.

3.3.2. The Main Weakness: User Reliance

By depending on the end user to directly allow or deny USB devices, we are assuming that they are fully aware of the dangers and risks that BadUSB holds; some studies have shown otherwise, where people tend to have higher tendencies to fall victim to curiosity and common interest in discovering the contents of USBs that are not secure.

As observed in (Tian et al., 2016), USB implementers direct the responsibility of avoiding malicious devices towards the end user, and as mentioned in the same research, users cannot verify the true functionality and genuine intent solely based on the external looks. They are more likely to plug in USB devices without being aware of the potential risks involved. Falsely assuming that the end user is aware of possibly malicious devices leads to significant security threats because the user's perception of most devices heavily depends on their external physical appearance, which leads to users believing that they are safer than they truly are (Kang et al., 2017).

User involvement in trust decisions is an error-prone and suboptimal design choice, especially when current countermeasures are insufficient and impose a burden on the end users to identify suspicious devices, which comes paired with a lack of appropriate trust modelling, and that is arguably not the best route for low-level system trust

decisions, especially when such interaction disrupts the primary task that the user has in hand (Neuner et al., 2018).

Malicious devices are often introduced to vulnerable systems due to users' mistakes (Fatos et al., 2016), and despite frequent warnings, recent studies showed that users are still prone to plugging in foreign devices without considering the hazards that are associated with unsafe USBs (Charan et al., 2023). UAC alerts were found annoying to the users, and the users instantly permitted execution (Pham et al., 2010). Researchers observed that users tend to dismiss popups cognitively (Bahr & Ford, 2011), especially when they get accustomed to the prompts; otherwise, they either uninstall or disable the security software (Mueller et al., 2019).

BadUSB attacks tend to exploit the weakest part of cyber defense, which is humans, proving they are the least proficient member (Arora et al., 2021), and we cannot assume that the common end user who has minimal technical background would treat a normal-looking USB device as a threat (Lu et al., 2021) since USB connections are mostly overlooked by both users and defenses as trusted and secure channels (Gates & Applied Computer Security Associates, 2010). Despite users being trained regularly to carefully inspect hardware for signs of tampering (Kierznowski & Mayes, 2016), user error and their vulnerability to social engineering make this process time-consuming and prone to mistakes, as shown in (Mueller et al., 2019) and (Blanchet, 2018; Tischer et al., 2016). When almost 300 infected USB flash drives were left lying in a company parking lot as part of an experiment, they monitored the number of connections that occurred using these devices by unsuspecting individuals. With high success rates, the first device was connected in less than six minutes, which indicates a lack of computer hygiene and proper security measures.

Numerous warnings do show the dangers that coexist with the use of USBs. Researchers demonstrated that users are still vulnerable to USB attacks (Charan & Kulkarni, 2022), yet a whole line of research still suggests involving the users in authorizing devices (Mueller et al., 2019) that includes countermeasures such as Tian et al.'s work (Tian et al., 2015), which indicates the need to look into more approaches (Kierznowski & Mayes, 2016) that are user-friendly and less reliant on humans' error-prone behavior (Griscioli et al., 2016).

4. DUCKTAPE: A NEW COUNTERMEASURE AGAINST USB ATTACKS

To tackle BadUSB threats, through the development of DuckTape those are the main points that are taken into consideration:

The most commercially available BadUSB attack is keystroke injection, with libraries that include ducky scripts and ease of access to reprogrammable microcontrollers.

The most used desktop operating system that is vulnerable to this exploit is Windows with the widest user base, and is observed to be most targeted in the mentioned injection libraries, as visualized in Figure 2.

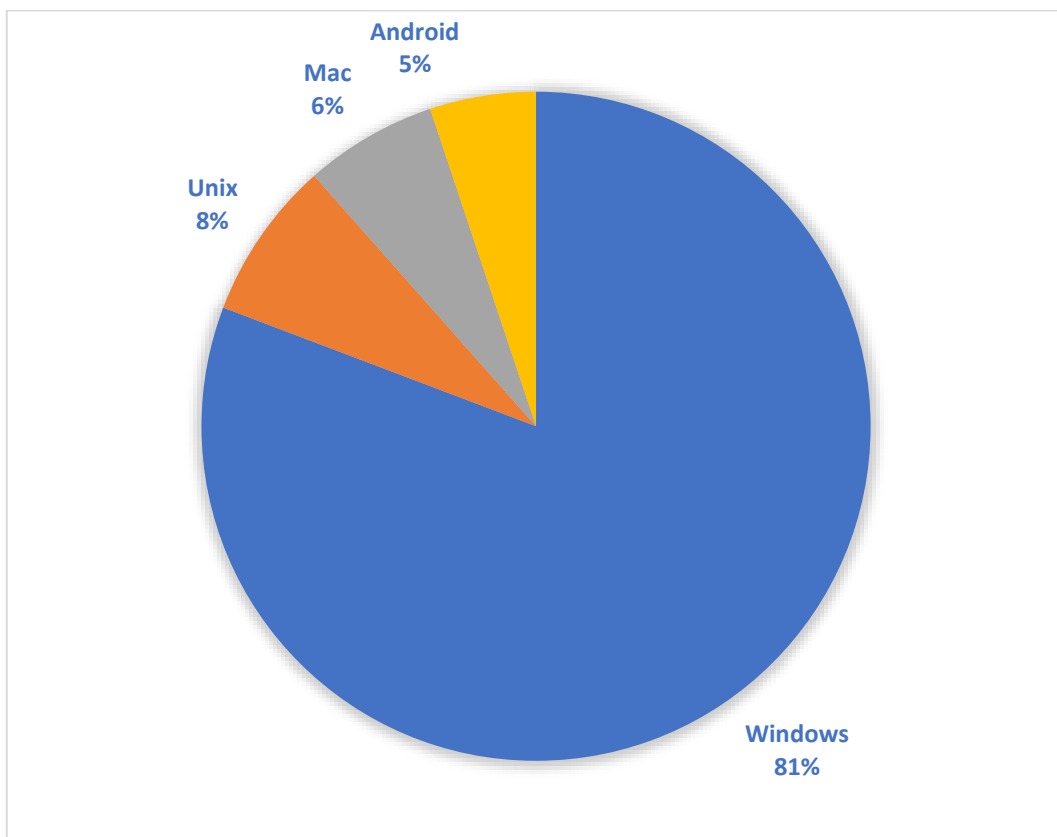


Figure 2 Targeted Operating Systems

The first instruction in most ducky scripts observed against windows machines is “GUI + R” with the purpose of accessing cmd or PowerShell, as shown in Figure 3.

To avoid mentioned weaknesses such as privacy risks and methods that are already bypassed such as whitelisting

The weakest link found in the mentioned security measures is usually the end user, that is in their behavior or interaction with the proposed countermeasure. Raising awareness is expected to play a significant role in reducing the risk of such attacks.

BadUSB attacks target different layers and are not just a singular form of attack, and can be combined to bypass proposed measures.

The usability and functionality of USB are both crucial and are not to be impacted severely by the countermeasure. Specific approaches have their advantages and are found lacking in distinct stages.

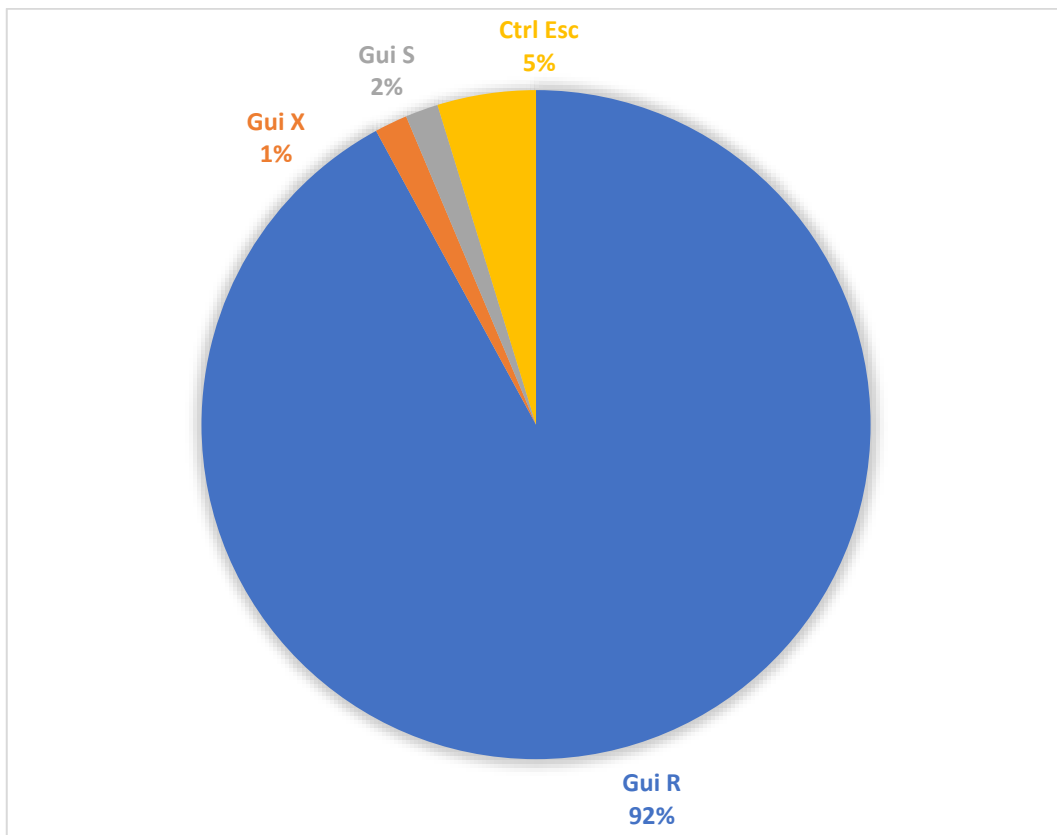


Figure 3 Initial Ducky Script Instruction - After Delay

4.1 Overview

DuckTape is a simple yet powerful tool designed to provide robust protection against BadUSB keystroke injection attacks. It accomplishes this by implementing a multi-layered defense strategy that encompasses distrust of all USB devices, hooking potentially dangerous keystrokes, and introducing various obstacles to hinder the

success of attacks. The primary objective of DuckTape is to ensure user safety without compromising privacy or introducing error-prone approaches. It achieves this by maintaining the normal functionality of USB devices for intended everyday use, as opposed to employing extreme measures that grant all-or-nothing access.

DuckTape is built on the principles of minimizing the attack surface while considering the most common threat models and scenarios. It takes a proactive approach, mitigating the impact of attacks rather than attempting to eliminate them entirely. This is achieved through the concept of zero trust and privilege de-escalation, coupled with optional limitations that empower users to tailor their security level according to their individual preferences. By striking a balance between security and usability, DuckTape ensures that the user experience remains largely unchanged, with only minor adjustments required to enhance system protection.

Although DuckTape primarily targets the most prevalent operating system, which is Windows, its underlying concept is applicable to other systems as well. This flexibility arises from the fact that DuckTape is not limited to specific functionalities or dependencies, making it adaptable across various platforms.

4.2 Key Components and Implementation Details

DuckTape is designed to be easily deployable without requiring any hardware modifications. It consists of a single executable application that occupies minimal storage space. The main menu of DuckTape that is displayed in Figure 4, provides users with a range of toggleable functionalities. These include:

List Button: Displays information about currently plugged-in USB devices, allowing users to identify potential threats.

Secure UAC Button: Modifies the system registry to prompt the user for their password whenever attempting to open the terminal as an administrator or access other high-privilege components. This helps prevent unauthorized access to critical system functions.

AutoStart Toggle: Adds DuckTape to the system's startup list, ensuring immediate protection upon system boot.

Disable Winkey Toggle: Hooks the super key (Windows key) that opens the start menu. This action partially prevents most Ducky scripts, which commonly utilize this key as the first step in an attack, from executing their intended commands.

Policy Editor Guide: Launches the Local Group Policy Editor and guides users to access the Windows key hotkey option and disable it, preventing Windows key combinations such as Win+R (opening the Run window), which are frequently targeted by Ducky scripts.

Mouse Only Checkbox: Modifies the verification process of the captcha window, making it clickable only using the mouse. By intercepting the Enter key from keyboard input, even attackers with keyboard emulation abilities cannot bypass the captcha window, even if they possess the correct salt.

reDucktion Button: Enables all the aforementioned functions together, instantly activating full protection against BadUSB attacks.

Additionally, DuckTape features a Captcha Length Slider that allows users to adjust the length of the captcha, providing flexibility and adaptability to different security needs.

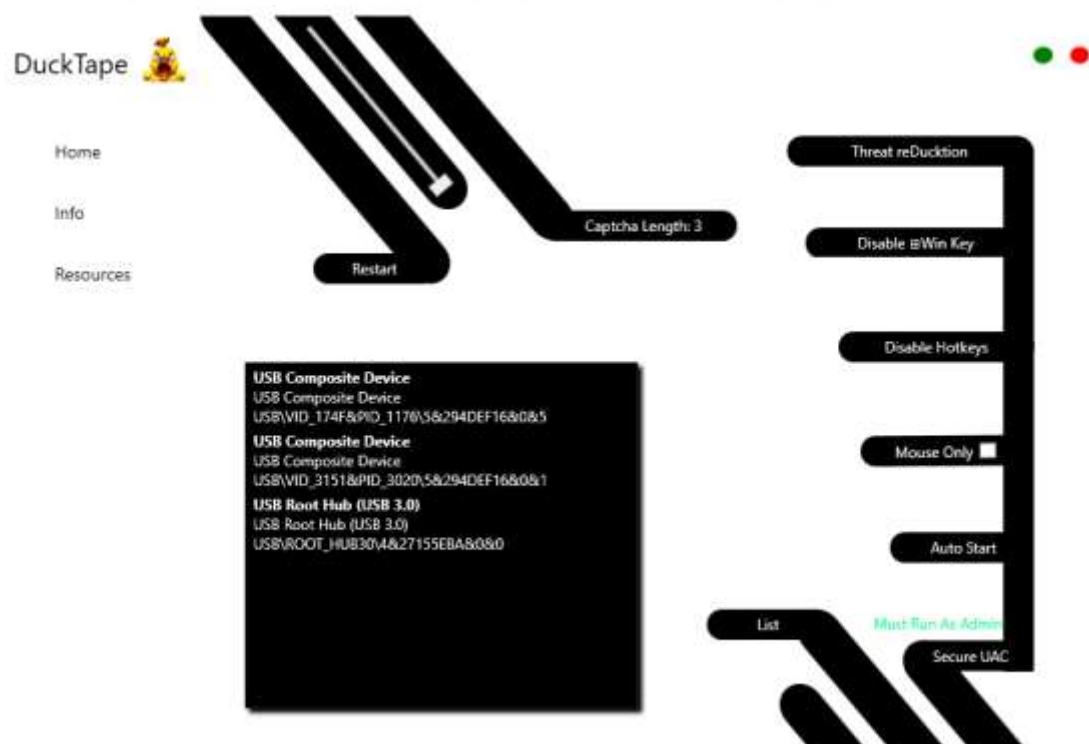


Figure 4 DuckTape GUI Main Menu

Upon the insertion of a new USB device, DuckTape opens a captcha window, as seen in Figure 5, that cannot be closed unless the captcha is entered correctly alongside the user-defined salt. During this process, all keys other than captcha related ones are hooked, restricting access to critical components like the start menu. In the event of an actual attack, DuckTape displays all attempted letter keystrokes to the end user, offering the ability to log and keep a record of these attempts. This feature creates a simulated honeypot environment without compromising user privacy. The application utilizes two text files to support specific functionalities: one for storing the salt value and another for managing the length of the captcha.



Figure 5 DuckTape GUI Captcha Window

DuckTape is primarily developed using the C# programming language within a WPF application on the .NET framework. It should be noted that the secure UAC functionality requires running the application as an administrator to ensure effective execution.

4.3 Usability and Complexity:

This tool is specifically developed for the Windows 10 operating system and has been tested for compatibility with Windows 11. The application is designed to offer a user-friendly experience while considering the complexity of its functionalities. All the functions presented on the main menu of DuckTape are optional, allowing users to customize their level of protection based on their individual needs. Depending on the user's requirements and preferences, they have the flexibility to enable or disable

specific functions. This approach ensures that DuckTape does not impose unnecessary restrictions on the usability of the host system.

It's important to note that even if users choose not to utilize certain functionalities, DuckTape employs a zero-trust approach when detecting and handling newly plugged devices. This means that all newly inserted USB devices are treated as potential threats, regardless of the functions enabled or disabled by the user. This proactive stance helps to minimize the attack surface and maintain a robust security posture.

4.4 Strengths and Advantages:

This solution offers several strengths and advantages in its approach to mitigating attacks and enhancing security:

Multi-Layered Protection

The countermeasure implements a series of layered protection mechanisms to defend against various attack vectors. At the beginning, a captcha is required to prevent automatic offsite keystroke injection attacks. In addition, an onsite threat actor is deterred by the utilization of a salt that accompanies the captcha, rendering the inescapable window difficult to bypass. The keyboard hook feature ensures that sensitive keystrokes cannot be injected to evade the captcha window. Furthermore, access control to sensitive privileged software is enforced through User Account Control (UAC), adding an extra layer of defense. Attack interrupts are also implemented by disabling specific functions commonly exploited by malicious scripts that assume the default settings of a system.

Customizability and Flexibility

The proposed approach enhances efficiency and security by allowing users to determine the length of the captcha. Users also have the ability to toggle specific features on or off based on their desired impact and frequency of use. This customization empowers users to achieve a balance between security and usability. The option to launch DuckTape with the auto-start menu ensures immediate protection for all sessions without requiring further interaction.

Enhanced Usability

Despite its robust security measures, DuckTape remains user-friendly with minimal interaction required. Even with a shorter captcha and a salt consisting of only one letter, brute-forcing becomes significantly more difficult especially with a regenerated captcha per attempt. The "reduction" button instantly enables all safety features at maximum level, providing comprehensive protection. Additionally, an automated button simplifies the process of securing UAC, making it more accessible to users with limited technical knowledge. For policy editing, a guiding message popup assists users in accessing and enabling the necessary settings.

Compatibility and Seamless Integration

The chosen framework is designed to work effectively even when the system is logged off or during the login process. Extensive testing has been conducted to ensure that it does not impact the logging in process or cause any disruptions. This compatibility ensures a seamless integration with the user's workflow without compromising security.

Comprehensive Defense at Different Attack Stages

The designed tool provides different layers of defense at various stages of an attack. By addressing identification, protection, detection, response, and recovery, it aligns with the National Institute of Standards and Technology (NIST) framework. This comprehensive approach enhances the overall security posture

Wide Applicability and User-Friendly Interface

Our approach is developed for the most common operating system, making it accessible to a broad range of users. It is designed to be usable by individuals across different proficiency levels, not limited to highly technical personnel or specific environments. Its user-friendly interface and clear functionality enable users to adopt and implement the tool easily.

Open Source and Community Contribution

This software is an open-source project, allowing contributions from the security community. This collaborative approach fosters continuous improvement, innovation, and peer review, ultimately strengthening the tool's effectiveness and reliability.

Cost-Effective Solution

This tool offers a free option, eliminating implementation costs for large enterprises and organizations. This affordability makes it an attractive choice for entities looking to enhance their security measures without significant financial investments.

Zero Trust Approach

The implemented measure adopts a zero trust model by distrusting all devices and considering them potential threats. This approach counteracts masquerading and mimicking techniques used by attackers to appear as normal behavior. By applying zero trust principles, DuckTape serves as a significant deterrent to attackers, hindering their operations and forcing them to consider alternative methods.

4.5 Limitations:

Vulnerability to Disabling: DuckTape can be turned off either by an attacker or by the user themselves. This limitation implies that if an attacker gains access to the system or the user disables DuckTape, the protection it provides will be circumvented. It is essential to educate users about the importance of keeping DuckTape enabled and regularly updated.

Lack of Detection Method After Authentication: DuckTape currently does not implement a detection method after the initial authentication process. Once a user enters the correct captcha and salt, DuckTape assumes the device is safe as long as it stays plugged to the host. However, future enhancements could include continuous monitoring or dynamic detection techniques to identify potential malicious behavior even after authentication.

Potential Lack of Future Proofing: There is a possibility that attackers may identify the protection mechanism and acquire the user set salt before launching an attack or just assuming the default value of it remains unchanged. This limitation highlights the importance of continuous improvement and adapting DuckTape to stay ahead of evolving attack methods. Future iterations could explore more secure mechanisms for captcha generation and salt implementation.

Reliance on Text Files for Settings: While using .txt files serves as a functional proof of concept, it may not be the most professional or robust method for storing and managing

DuckTape settings. Consider exploring alternative, more secure approaches for storing and accessing configuration settings to ensure better usability and maintainability.

Hardware-Focused Attacks: DuckTape primarily focuses on software-side protection and may offer limited defence against attacks targeting hardware vulnerabilities. Recognizing that the effectiveness of DuckTape is primarily within the software domain and the importance of implementing complementary hardware-level security measures for specific environments is always an option.

Compatibility with Unique USB Drivers: The adopted system may encounter compatibility issues with certain USB devices that require specific or proprietary drivers. These drivers might introduce overlays or interfere with the functionality of DuckTape's captcha window. As a potential workaround or considerations to ensure compatibility with a wider range of USB devices, DuckTape captcha window interface is forced to display over all other windows by upholding focus on the interface every 500 millisecond.

Limited Protection against Advanced Attacks: While Our technique provides robust defence against common keystroke injection attacks, it is important to acknowledge that determined and sophisticated attackers may still find ways to circumvent the countermeasure. This strategy aims to mitigate the majority of known attacks but cannot guarantee protection against all possible attack vectors. Continuous testing, updates, and proactive security measures are essential to address emerging threats.

4.6 Testing

In order to comprehensively evaluate the effectiveness of DuckTape against BadUSB, a testing environment and appropriate equipment are essential. This chapter outlines the hardware and software components, necessary to conduct the research and validate the proposed countermeasure.

4.6.1. Testing Environment Setup:

4.6.1.1 Hardware Setup:

- Raspberry Pi Pico: microcontroller board as the Rubber Ducky alternative.

- Test device: common brand personal computer with default settings, USB ports.

4.6.1.2 Software Setup:

- Operating system: Windows 10, Default protection settings, latest update 2023.
- CircuitPython: programming language to code on microcontroller boards.
- Adafruit bundle: containing library that allows Pico to act as HID device.
- Duckyinpython.py: converts ducky script to be used in python
- Payloads: pre defined keystrokes written in ducky script.
- Novabench 5: to help understand the approximate performance impact caused by DuckTape

4.6.2. Testing Methodology:

4.6.2.1 Test Scenarios and Use Cases:

- Visual Ambiguity, this is not the only way threat actors can utilize stealth in their attacks, it is possible to run a VBS script that runs hidden from the users reach (adamffl, n.d.) in this test we are running (NaddSafety, n.d.), this ducky script as seen in Figure 6, with the results shown in Figure 7. The interception is presented in Figure 8.

```
GUI r
DELAY 1000
STRING powershell.exe
ENTER
DELAY 200
STRING $host.ui.rawui.windowSize = New-Object System.Management.Automation.Host.Size 1, 1
DELAY 100
ENTER
```

Figure 6 DuckyScript - VisualAmbiguity



Figure 7 Attack Result



Figure 8 DuckTape Interception

- Assuming that the threat actor somehow could obtain both the salt and the captcha correctly and was able to breach the first layer of defence, and implemented enough delay to manually bypass the captcha window without interrupting the attack. then attempted to turn off windows defender real-time protection as evidenced in Figure 9:



Figure 9 DuckyScript Disable Defender and Attack Result

After activating UAC protection function, even if the threat actor obtained the salt and correctly entered the captcha, which is already a hindrance and significant delay, User Account Controller demands for the password of the device, this comparison is represented in Figure 10, demonstrating the strength of multilayered protection.



Figure 10 UAC - Bypassable (Left) - Secure (Right)

Although, it does block access to terminal and shell with admin privileges, but still allows for usage with the user privilege which can be exploited to elevate privileges but this can still be enough of a deterrence and reduces the potential risk margin.

4.6.2.2 Test Metrics and Measurements:

- Detection rates of DuckTape were 100% all the time while testing, it treats any new device as a threat, even upon renumeration, which was tested by switching between different modes of USB usage without unplugging the phone and reintroducing it to the

system using the USB settings menu shown in Figure 11. DuckTape treated every change as a threat and interrupted the interface with the Captcha window every time.



Figure 11 USB Settings - Android

AutoPlay

Memory card



Figure 12 AutoPlay Settings - Windows 10

- Since AutoPlay feature defaulted for the device to view files using file explorer as indicated in Figure 12, the interface of File explorer kept appearing on top of the captcha window and interrupting the verification process and the keyhooks as well, this was dealt with by introducing a loop that makes sure that captcha window reappears on top every 500 millisecond, the code snippet is shown in Figure 13.

```
1 reference
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    GenerateCaptcha();
    textBox1.Focus();

    foregroundTimer = new System.Timers.Timer(500);
    foregroundTimer.Elapsed += ForegroundTimer_Elapsed;
    foregroundTimer.Start();
}

1 reference
private void ForegroundTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    Dispatcher.Invoke(() =>
    {
        SetForegroundWindow(new System.Windows.Interop.WindowInteropHelper(this).Handle);
    });
}

private System.Timers.Timer foregroundTimer;
```

Figure 13 Interface Overlapping Solution

4.6.3 Testing Limitations:

- Testing environment and equipment were limited, such as the inability to test keyboard LED attacks or multilayered attacks and the use of sophisticated scripts for exfiltration via E-mail, Dropbox, or Discord. although exfiltration methods on the ducky itself were demonstrated, and defended against.

- Some exfiltration methods that do not need admin privilege can still successfully occur against DuckTape if the threat actor managed to bypass the captcha window by social engineering the victim and having the injection delayed until after the verification process is over.

- If the attacker turned off DuckTape before executing malice, and the user never turned on any secondary functions to enhance defenses against injection attacks, such as UAC or Disabling Hotkeys, all security measures are successfully avoided and scripts will run as expected by the attacker.

4.6.4 Evaluation:

In terms of performance overhead, the tested device appeared unphased when idling DuckTape in the background and when it was benchmarked, overall score was impacted by 0.0856% as the difference is pointed observed in Figure 15, perhaps more advanced performance testing can reveal the exact amount of resources that are being consumed by DuckTape in future work.



Figure 15 System Benchmark - DuckTape Idling

- Through security testing over the suggested

```
1 DELAY 100
2 APP
3 DELAY 100
4 STRING W
5 DELAY 100
6 STRING T
7 DELAY 100
8 ENTER
9 DELAY 100
10 ENTER
11 ENTER
12 DELAY 1000
13 STRING start powershell
14 DELAY 200
15 CTRL SHIFT S
16 DELAY 200
17 STRING zpwrschl.bat
18 ENTER
19 DELAY 100
20 ENTER
21 DELAY 100
22 ALT F4
23 DELAY 300
24 RIGHTARROW
25 ENTER
26 DELAY 300
27 TAB
28 TAB
29 TAB
30 TAB
31 RIGHTARROW
32 RIGHTARROW
33 DELAY 400
34 ENTER
35 DELAY 400
36 STRING z
37 STRING z
38 ENTER
```

Figure 14 Bypassing Windows Key and Hotkeys to access PowerShell

features in DuckTape, “Disabling Windows Key Hotkeys” using local group policy editor is proven less reliable because it is still possible to access the terminal without using Windows key or hotkeys at all, as presented in this DuckyScript in Figure 14, it is shown that the threat actor can create a new batch file on the desktop and access it to trigger PowerShell window. Even though this is not the common way used to reach PowerShell, usually scripts use Win+R to reach terminal GUI, the use of Batch files or the key combination of Ctrl+Esc, Win+X is still an option even if the use of Win+R, Win+S are disabled. That is still a secondary layer of defence and a welcomed feature, and can only be exploited if the captcha window that is the main trap was completely bypassed, exposing multiple points of resistance against the attack.



5. DISCUSSION

DuckTape incorporates a very basic methods of verification, isolation, logging, privilege de-escalation, optional elimination of unnecessary functions. all while practicing safe approach that does not negatively impact the user experience.

Yet it encourages users of all technical abilities to exercise caution with foreign devices without using excessive mechanisms that either limit the functionality of a system or excessive resource usage.

By analyzing the attacks library in Hak5 GitHub, it was found that these attacks rely on default system settings. Their strength points were speed, stealth, and user privilege.

DuckTape is designed to hinder speed by introducing different obstacles against these attacks, the stealth factor is exposed since the user must interact with a screen that cannot be closed without understanding the introduction of a new device and warning the user to remove it as soon as possible.

Privileges can collide with enterprise policies; this is one of the secondary functions that can be controlled depending on the users' desires.

6. CONCLUSIONS

Specifically, this work's contribution is summarized as follows:

It is identified that the major vulnerabilities lying under the misuse of USB devices and the failure of systems to prevent BadUSB attacks when granting instant full trust as a byproduct of physical proximity to the host.

Possible exploits accompanied by the faulty protocol are acknowledged which include vulnerable drivers, keystroke injections, and eavesdropping, and it is pointed out hypothetical possible candidate scenarios for works to explore proper defenses against; those can only expand further as hosts and operating systems remain carrying the same design of trust for connected devices using USB ports.

The commonly overlooked security risks in recent research and specifications that indicate the lack of consideration when applying some mechanisms are highlighted, and understanding that due to multiple layers of the attack, one layer of security cannot fully provide prevention without massive compromises to usability and introducing restrictions or a safety gap, which allowed this research to be more aware of the downsides of other countermeasures.

The behavior characteristics of injection attacks based on an openly available extensive library are studied and an appropriate countermeasure is designed that can be implemented by the widest userbase on the most popular operating system to effectively minimize the threat surface while increasing user awareness of the threat.

DuckTape provides end users and security conscious environments with a prevention mechanism against keystroke injections. It helps with authorizing devices introduced to the host USB port while understanding the balance between security and usability without posing further risks to the users privacy or majorly effecting the user experience.

It enforces zero trust policy over the USB protocol and encourages users to toggle specific functionalities and features that they might not use which are usually exploited by the threat actor.

DuckTape is suitable for currently defenseless systems to mitigate off the shelf and somewhat advanced threats by eliminating the instant trust approach, therefore maximizing protection for most users and possible scenarios. By applying a valid

simple and effective protection with low overhead that suits implementation within different operating systems and can be placed at security environments without major alteration to the existing systems, to offer access control, privilege limiting, and overall weakening the strengths of injection attacks.

DuckTape functions with high levels of effectiveness beside optional access limitations that could not impact the average user, which indicates the flexibility among implementation and different protection methods. Yet it holds some limitations too and is not considered future proof whatsoever, with no strong guarantees of attack elimination, only delivering pure mitigation tactics to minimize the threat surface.

Assuming the user introduced the changes to the system settings that are applied even if DuckTape was disabled or tricked to thinking that a new device was not plugged, with hopes that the proposed measure is found useful and facilitated on multiple platforms that aim for enhancing defense methods.

This open source project allows contributions from the field to further improve the possible abilities of DuckTape to spread awareness and effective protection methods for all users, mostly the outlined contribution of DuckTape is architectural since some of the features are enhanced from prior works and existing protection methods.

The exposure of BadUSB exploits helped in understanding the resulting vulnerabilities and further comprehending that this widespread cross-platform misconfigured protocol cannot be protected with a single countermeasure that is comprehensive and effective while considering usability and cost. Since this attack comes frequently coupled with multiple threats on more than just one layer of the host architecture, it was not feasible to include protection against all different variants of BadUSB attacks in DuckTape. Some countermeasures and tools fell out of scope and deserve further investigation in future work and are worth looking into, such such as UScramBLE, Ghost USB HoneyPot, PinUP, ELinux, SandUSB, USBProxy, Rushby separation kernel, USBSec, USBGuard, USG, TypingDNA, KeyTrac, Digital Guardian, USB Proxy, Malboard.

7. RESOURCES

- adamff1. (n.d.). *Rickroll MODDED*. Retrieved July 12, 2023, from https://github.com/adamff1/Rickroll_MODDED_HID
- Angel, S., Wahby, R., Leners, J., & Blumberg, A. (2016). *Defending against Malicious Peripherals with Cinch*. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_angel.pdf
- Arora, L., Thakur, N., & Yadav, S. K. (2021). USB rubber ducky detection by using heuristic rules. *Proceedings - IEEE 2021 International Conference on Computing, Communication, and Intelligent Systems, ICCICIS 2021*, 156–160. <https://doi.org/10.1109/ICCICIS51004.2021.9397064>
- Baezner, M., & Robin, P. (2017, October). *Stuxnet*. <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/200661/Cyber-Reports-2017-04.pdf?sequence=1&isAllowed=y>
- Bahr, G. S., & Ford, R. A. (2011). How and why pop-ups don't work: Pop-up prompted eye movements, user affect and decision making. *Computers in Human Behavior*, 27(2), 776–783. <https://doi.org/10.1016/j.chb.2010.10.030>
- Blanchet, S. (2018). *BadUSB, the threat hidden in ordinary objects*. https://www.researchgate.net/publication/331876425_BadUSB_the_threat_hidden_in_ordinary_objects
- Cannols, B., & Ghafarian, A. (2017). *Hacking Experiment by Using USB Rubber Ducky Scripting*. <https://www.iiisci.org/journal/pdv/sci/pdfs/ZA340MX17.pdf>
- Charan, V., & Kulkarni, L. (2022). Enhancement And Implementation Of Badusb Attacks Using Microcontrollers. *Journal of Positive School Psychology*, 563–573. [https://doi.org/10.1109/SP.2018.00037.\[7](https://doi.org/10.1109/SP.2018.00037.[7)
- Charan, V., Kulkarni, L., & Karad, V. (2023). Survey On Micro-Controller Based Bad USB Attacks. In *Journal of Positive School Psychology* (Vol. 2023, Issue 1). <https://www.journalppw.com/index.php/jpsp/article/view/15319/9872>
- Clark, J., Leblanc, S., & Knight, S. (2011). Compromise through USB-based Hardware Trojan Horse device. *Future Generation Computer Systems*, 27(5), 555–563. <https://doi.org/10.1016/j.future.2010.04.008>
- Common Vulnerabilities and Exposures*. (n.d.). Retrieved July 12, 2023, from <https://cve.mitre.org>

- dallaswinger. (n.d.). *Jitter*. Retrieved July 12, 2023, from <https://github.com/hak5/usbrubberducky-payloads/tree/master/payloads/examples/Jitter>
- Denney, K., Erdin, E., Babun, L., Vai, M., & Uluagac, S. (2019). USB-watch: A dynamic hardware-assisted USB threat detection framework. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, 304 LNICST*, 126–146. https://doi.org/10.1007/978-3-030-37228-6_7
- Dumitru, R., Wabnitz, A., Genkin, D., & Yarom, Y. (2022). *The Impostor Among US(B): Off-Path Injection Attacks on USB Communications*. https://www.usenix.org/system/files/sec23summer_9-dumitru-prepub.pdf
- Farhi, N., Nissim, N., & Elovici, Y. (2019). Malboard: A novel user keystroke impersonation attack and trusted detection framework based on side-channel analysis. *Computers and Security*, 85, 240–269. <https://doi.org/10.1016/j.cose.2019.05.008>
- Fatos, L. B., Kangbin, X., & Editors, Y. (2016). *Advances on Broad-Band Wireless Computing, Communication and Applications Lecture Notes on Data Engineering and Communications Technologies*. <http://www.springer.com/series/15362>
- Gates, Carrie., & Applied Computer Security Associates. (2010). *Exploiting smart-phone USB connectivity for fun and profit*. <https://dl.acm.org/doi/abs/10.1145/1920261.1920314>
- Gillespie, J. (n.d.). *IronKey*. Retrieved July 12, 2023, from <https://www.businesswire.com/news/home/20140808005723/en/IronKey™-Secure-USB-Devices-Protect-Against-BadUSB-Malware>
- Google. (n.d.-a). *UKIP*. Retrieved July 12, 2023, from <https://github.com/google/ukip>
- Google. (n.d.-b). *UKIP Google*. Retrieved July 12, 2023, from <https://opensource.googleblog.com/2020/03/usb-keystroke-injection-protection.html>
- Griscioli, F., Pizzonia, M., & Sacchetti, M. (2016). *USBCheckIn: Preventing BadUSB Attacks by Forcing Human-Device Interaction*. <https://doi.org/10.1109/PST.2016.7907004>
- Hak5. (n.d.-a). *Attack Modes - USB Rubber Ducky*. Retrieved July 12, 2023, from <https://docs.hak5.org/hak5-usb-rubber-ducky/attack-modes-constants-and-variables/attack-modes>
- Hak5. (n.d.-b). *Jitter Hak5*. Retrieved July 12, 2023, from <https://docs.hak5.org/hak5-usb-rubber-ducky/advanced-features/jitter>
- Hak5Darren. (n.d.). *The Official USB Rubber Ducky Payload Repository*. Retrieved July 12, 2023, from <https://github.com/hak5/usbrubberducky-payloads/tree/master/payloads/library>

- Hammoudeh, M., Watters, P., Epiphaniou, G., Kayes, A. S. M., & Pinto, P. (2021). Special issue “security threats and countermeasures in cyber-physical systems.” In *Journal of Sensor and Actuator Networks* (Vol. 10, Issue 3). MDPI AG.
<https://doi.org/10.3390/jsan10030054>
- Kang, M.-G., Luo, B., & Li, F. (2017). *USBWall: A Novel Security Mechanism to Protect Against Maliciously Reprogrammed USB Devices*.
<https://www.tandfonline.com/doi/full/10.1080/19393555.2017.1329461>.
- Karystinos, E., Andreatos Antonios, & Douligieris Christos. (2019). *Spyduino: Arduino as a HID Exploiting the BadUSB Vulnerability*. <https://doi.org/10.1109/DCOSS.2019.00066>
- KasperSky. (n.d.). *Change Kaspersky settings to allow USB devices*. KasperSky. Retrieved July 12, 2023, from <https://it.nmu.edu/docs/change-kaspersky-settings-allow-usb-devices>
- Kharraz, A., Daley, B. L., Baker, G. Z., Robertson, W., & Kirda, E. (2019). USBESAFE: An End-Point Solution to Protect Against USB-Based Attacks. *USENIX Association 22nd International Symposium on Research in Attacks, Intrusions and Defenses*, 89–103.
https://www.usenix.org/system/files/raid2019-kharraz_0.pdf
- Kierznowski, D., & Mayes, K. (2016). *BADUSB 2.0: EXPLORING USB MAN-IN-THE-MIDDLE ATTACKS*. http://docs.media.bitpipe.com/io_10x/io_102267/item_1306461/RH-2016-BadUSB-DavidKierznowski.pdf
- Lu, H., Wu, Y., Li, S., Lin, Y., Zhang, C., & Zhang, F. (2021). *BADUSB-C: Revisiting BadUSB with Type-C*.
- Mohammadmoradi, H., & Gnawali, O. (2018). Making whitelisting-based defense work against bad USB. *ACM International Conference Proceeding Series*, 127–134.
<https://doi.org/10.1145/3289100.3289121>
- Mueller, T., Zimmer, E., & De Nittis, L. (2019, August 26). Using context and provenance to defend against USB-borne attacks. *ACM International Conference Proceeding Series*.
<https://doi.org/10.1145/3339252.3339268>
- NaddSafety. (n.d.). *Resize*. Retrieved July 12, 2023, from
<https://github.com/NaddSafety/busb/blob/main/resize%20to%20small.txt>
- Neuner, S., Voyiatzis, A. G., Fotopoulos, S., Mulliner, C., & Weippl, E. R. (2018). *USBlock: Blocking USB-based Keypress Injection Attacks*.
https://link.springer.com/chapter/10.1007/978-3-319-95729-6_18

- Nicho, M., & Sabry, I. (2022). Threat and Vulnerability Modelling of Malicious Human Interface Devices. *Technology, Engineering & Mathematics (EPSTEM)*, 21. www.isres.org
- Nissim, N., Yahalom, R., & Elovici, Y. (2017). USB-based attacks. In *Computers and Security* (Vol. 70, pp. 675–688). Elsevier Ltd. <https://doi.org/10.1016/j.cose.2017.08.002>
- Nohl, K., Krißler, S., & Lell, J. (2014). *BadUSB-On accessories that turn evil*.
- Pham, D. V., Halgamuge, M. N., Syed, A., & Mendis, P. (2010). *Optimizing Windows Security Features to Block Malware and Hack Tools on USB Storage Devices*.
- pmsosa. (n.d.). *DuckHunt*. Retrieved July 12, 2023, from <https://github.com/pmsosa/duckhunt>
- Rankin, K. (2009). *When Disaster Strikes: Attack of the rm Command*. <https://www.linuxjournal.com/article/10428>
- Sikka, S., Srivastva, U., Sharma, R., & Student, M. T. (2017). A Review of Detection of USB Malware. In *International Journal of Engineering Science and Computing*. <http://ijesc.org/>
- Takehisa, T., Iwamura, M., & Ushimaru, H. (2017). *Studies on Countermeasures for Malicious USB Devices*. <https://www.nict.go.jp/publication/shuppan/kihou-journal/journal-vol63no2/journal-vol63no2-05-04.pdf>
- Tian, D., Bates, A., & Butler, K. (2015). Defending against malicious USB firmware with GoodUSB. *ACM International Conference Proceeding Series, 7-11-December-2015*, 261–270. <https://doi.org/10.1145/2818000.2818040>
- Tian, D., Scafe, N., Bates, A., Butler, K., & Traynor, P. (2016). Making USB Great Again with USBFILTER. *Proceedings of the 25th USENIX Security Symposium*, 48. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_tian.pdf
- Tischer, M., Durumeric, Z., Foster, S., Duan, S., Mori, A., Bursztein, E., & Bailey, M. (2016). Users Really Do Plug in USB Drives They Find. *2016 IEEE Symposium on Security and Privacy (SP)*, 306–319. <https://doi.org/10.1109/SP.2016.26>
- usbkill. (n.d.). Retrieved July 12, 2023, from <https://usbkill.com>
- usbmon. (n.d.). The Kernel Development Community. Retrieved July 12, 2023, from <https://docs.kernel.org/usb/usbmon.html>
- Wrightson, T. (2014). *Advanced Persistent Threat Hacking: The Art and Science of Hacking Any Organization*. <https://dl.acm.org/doi/abs/10.5555/2823850>

- Yang, B., Qin, Y., Zhang, Y., Wang, W., & Feng, D. (2016). *TMSUI: A Trust Management Scheme of USB Storage Devices for Industrial Control Systems* (S. Qing, E. Okamoto, K. Kim, & D. Liu, Eds.; Vol. 9543). Springer International Publishing.
https://doi.org/https://doi.org/10.1007/978-3-319-29814-6_13
- Zuyeva, Ye. A., & Ordabayeva, G. K. (2022). RESEARCH OF THE SAFETY OF THE OPERATING SYSTEM WITH A MODIFIED USB-DEVICES. *Вестник Алматинского Университета Энергетики и Связи*, 4, 101–113. https://doi.org/10.51775/2790-0886_2022_59_4_101

