# An Intelligent USB Device Monitoring and Security Framework using Raspberry Pi and Ubuntu

Halel Itzhaki - 322989674

## 1. Introduction

USB devices have become an integral part of modern computing, serving as essential tools for data transfer, peripherals, and mobile connectivity. However, these devices also pose significant security risks, as they can act as vectors for malicious software or unauthorized access to sensitive systems. Traditional security measures, which often rely on static rule-based configurations, struggle to adapt to the dynamic nature of USB-based threats.

This project introduces an intelligent USB Device Monitoring and Management System utilizing Ubuntu's capabilities and a Raspberry Pi configured as a USB gadget. This dual-component system offers real-time monitoring, integrates a semi-supervised learning model, and adapts based on user interactions and new device behaviors, providing robust and adaptive USB security management.

The core components of the project include:

1. Ubuntu USB Device Manager: A dynamic monitoring and management system that detects, identifies, and controls USB devices on Ubuntu, learning from user input to automate future responses.
2. Raspberry Pi USB Gadget: Configured as a flexible USB gadget capable of simulating various device types, like USB mass-storage or Keyboard.

## 2. Related Work

Current approaches to USB security and device management include:

- Rule-Based Systems: Examples like `usbguard` implement static configurations to control USB devices. These solutions provide a baseline level of security but lack the flexibility needed to respond to evolving threats autonomously.
- Attack Simulation Devices: Hardware tools like USB Rubber Ducky and Flipper Zero simulate USB attacks such as keystroke injection. These are useful for testing static defenses but are not integrated into a dynamic and adaptive monitoring system. And are illegal in Israel.
- udev Configurations: The Linux `udev` system allows automatic responses based on USB events. While `udev` rules can automate some actions, they do not incorporate machine learning for intelligent decision-making or adaptation based on historical behavior.
- Difference from Existing Solutions:
  The project leverages `udev` to detect USB events and integrates it with a semi-supervised learning model that evolves based on user behavior, enhancing

adaptability and reducing manual intervention. By configuring the Raspberry Pi as a programmable USB gadget, the system can simulate a bypassing of USB devices by impersonating an allowed USB device.

# 3. Architecture

The architecture of the project is designed to ensure modularity and adaptability. It comprises two main subsystems:

**3.1** Ubuntu USB Device Manager:

This component operates as a background process in Ubuntu, continuously scanning for USB devices. It utilizes the Linux `udev` system to detect and respond to USB events, and it interfaces with a GUI to engage the user for decision-making when necessary.

Core Modules:

- Device Detection and Identification Module: This module interfaces with `udev` to monitor the `/dev/` directory for changes triggered by USB connections. When a new device is detected, the module retrieves metadata such as the vendor ID, product ID, and device class to classify the device. The integration with `udev` ensures immediate response to device events, enhancing efficiency and security.
- User Interaction Module: A GUI popup provides the user with options to allow or block the device. Users can opt to automatically approve future devices from the same vendor, reducing repetitive prompts and optimizing user experience.
- Learning Module: This module incorporates a semi-supervised learning algorithm that records user responses and device metadata, building a dataset over time. The system uses this data to predict user preferences and automate decisions for similar devices, thereby improving efficiency and reducing manual intervention.

**3.2** Raspberry Pi USB Gadget:

The Raspberry Pi 4 is configured using the Linux `gadget` framework and `configfs` to emulate various types of USB devices. Modifications are made to `/boot/firmware/config.txt` with parameters such as `dtoverlay=dwc2` and `dr_mode=peripheral` to ensure the Pi operates in peripheral mode when connected to a host.

The Pi can simulate HID devices (e.g., keyboards), mass storage devices, or composite devices. This capability allows for dynamic testing and validation of the Ubuntu USB Device Manager’s performance in different scenarios, including those involving potential malicious behavior.

USB Gadget Configuration Structure:
/sys/kernel/confid/usb_gadget/test_gadget

```
|-- bcdDevice
|-- bcdUSB
|-- bDeviceClass
|-- bDeviceProtocol
|-- bDeviceSubClass
|-- bMaxPacketSize0
|-- configs
|   `-- c.1
|       |-- bmAttributes
|       |-- mass_storage.usb0 -> ../../../../usb_gadget/test_gadget/functions/mass_storage.usb0
|       |-- MaxPower
|       `-- strings
|-- functions
|   `-- mass_storage.usb0
|       |-- lun.0
|       |   |-- cdrom
|       |   |-- file
|       |   |-- forced_eject
|       |   |-- inquiry_string
|       |   |-- nofua
|       |   |-- removable
|       |   `-- ro
|       `-- stall
|-- idProduct
|-- idVendor
|-- max_speed
|-- os_desc
|   |-- b_vendor_code
|   |-- qw_sign
|   `-- use
|-- strings
|   `-- 0x409
|       |-- manufacturer
|       |-- product
|       `-- serialnumber
|-- UDC
`-- webusb
    |-- bcdVersion
    |-- bVendorCode
    |-- landingPage
    `-- use
```

# 4. Research of USB Gadgets Using ConfigFS

ConfigFS Exploration:
The use of `configfs` on the Raspberry Pi is central to configuring USB gadgets. It allows the creation and management of USB device functions (e.g., HID or mass storage) through file system interfaces. By structuring gadgets in the manner explained earlier, the system dynamically sets up and modifies USB devices, making the Pi a powerful testbed for USB device behavior.

The flexibility of `configfs` enables testing multiple device types and combinations, verifying the USB Device Manager's capacity to handle complex, composite USB gadgets effectively.

# 5. Methods

**4.1** Device Monitoring Process:
The monitoring system is developed using a combination of C and Python, leveraging C for direct interaction with USB device nodes and Python for higher-level functionality, such as GUI management and machine learning integration.

Integration with udev:
`udev` rules are critical in ensuring that the system reacts immediately to USB events. The rules monitor for new devices, and upon detection, they pass relevant information to the USB Device Manager, which processes the data and triggers user interaction or automated responses as needed. This efficient setup minimizes latency and maximizes system responsiveness.

**4.2** Semi-Supervised Learning Model:
The learning model collects data based on user interactions, recording attributes like vendor IDs, product IDs, and user responses. The dataset grows as users interact with the system, allowing the model to learn and make accurate predictions about future USB devices.
Automatic Learning and Adaptation:
The model can automate decisions for frequently connected devices after several interactions, reducing manual input by up to 80% for recognized vendors or device types.

**4.3** Raspberry Pi USB Gadget Configuration:
The Raspberry Pi uses `configfs` to dynamically configure itself as different USB devices. The framework allows for the creation of composite devices, such as a HID and storage device, which tests the versatility and robustness of the Ubuntu USB Device Manager.

# 6. Results

**5.1** Ubuntu USB Device Manager Performance:
The system demonstrated high efficiency in detecting and managing USB devices, with the integration of `udev` enabling prompt responses to device events. The GUI provided clear and intuitive user interaction, and the learning model accurately predicted user preferences for known devices.

**5.2** udev's Role and Impact:
`udev` allowed the system to respond to device changes in real-time, processing USB events immediately and updating the system's database of device actions accordingly. The `udev` rules were fine-tuned to optimize performance and minimize unnecessary alerts.

**5.3** Raspberry Pi USB Gadget Testing:
Initial testing showed that when the Raspberry Pi was connected through a USB hub, the system rejected the hub before the gadget was plugged in, preventing recognition. However, once the USB hub was approved, the Raspberry Pi connected successfully when it faked a vendor ID already present in the learning model's dataset, demonstrating the model's ability to correctly identify and approve devices based on learned preferences.

**5.4** Automatic Learning Outcome:
The system successfully automated responses for similar devices after learning from user interactions. Over several iterations, the model's accuracy increased, reducing manual approvals significantly for recognized device types and vendors.

# 7. References


1. https://en.wikipedia.org/wiki/Udev
2. https://stackoverflow.com/questions/39579727/obtain-device-path-from-pyudev-with-python
3. https://blog.stolle.email/2020/04/16/disable-automount-for-specific-usb-devices-in-linux/
4. https://www.gtk.org/docs/language-bindings/python
5. https://unix.stackexchange.com/questions/63199/how-to-disable-usb-devices-based-on-vendor-id-in-linux-environment
6. https://askubuntu.com/questions/1087057/i-want-to-block-all-usb-devices-except-usb-keyboard-and-usb-mouse-using-udev-rul
7. https://ieeexplore.ieee.org/abstract/document/10102264
   https://en.wikipedia.org/wiki/Security-Enhanced_Linux
8. https://www.redhat.com/en/topics/linux/what-is-selinux
9. https://usbguard.github.io
10. https://wiki.archlinux.org/title/USBGuard
11. https://chatgpt.com/

12. https://www.reddit.com/r/raspberry_pi/comments/10n1mri/is_there_any_way_to_make_a_pi_4_look_like_a_usb/?rdt=56115

13. https://github.com/dummy-decoy/flipperzero_badusb_kl/blob/master/main.c
14. https://github.com/kmpm/rpi-usb-gadget.git
15. https://github.com/techcraftco/rpi-usb-gadget.git
16. https://blog.hardill.me.uk/2019/11/02/pi4-usb-c-gadget/
17. https://raspberrypi.stackexchange.com/questions/32197/can-the-pi-emulate-an-hid-device-with-via-usb
18. https://learn.adafruit.com/turning-your-raspberry-pi-zero-into-a-usb-gadget/ip-addressing-options
19. https://www.youtube.com/watch?v=34XnnAFO8O0
20. https://docs.kernel.org/usb/gadget_configfs.html
21. https://unix.stackexchange.com/questions/427161/binding-linux-gadget-to-udc-driver
22. https://www.collabora.com/news-and-blog/blog/2019/02/18/modern-usb-gadget-on-linux-and-how-to-integrate-it-with-systemd-part-1/
23. https://github.com/qlyoung/keyboard-gadget/blob/master/gadget-setup.sh