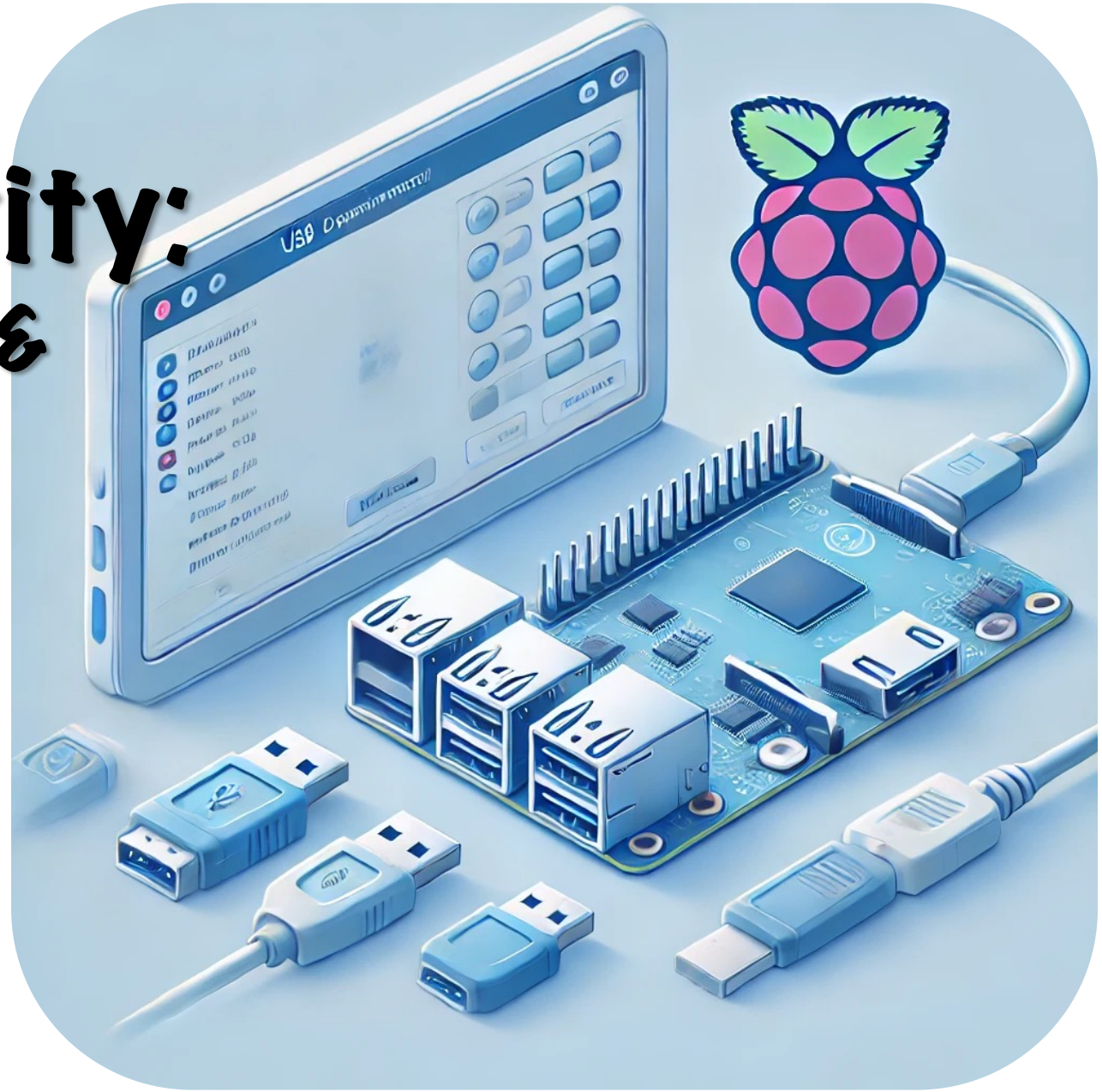
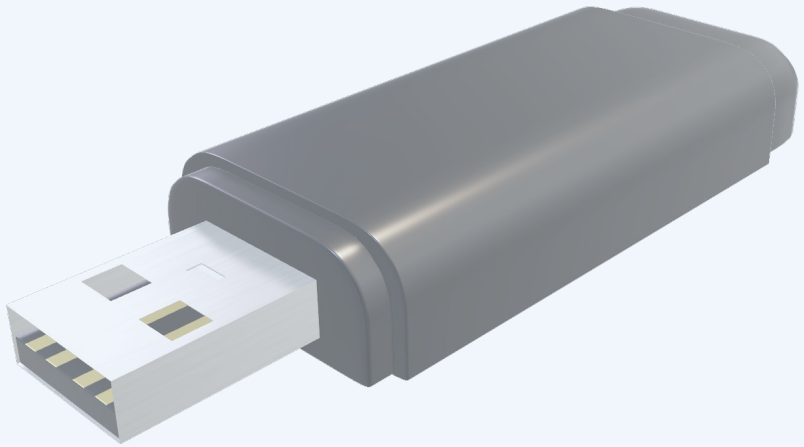
USB Device Security:
An Ubuntu USB Manager & Malicious Gadget Study



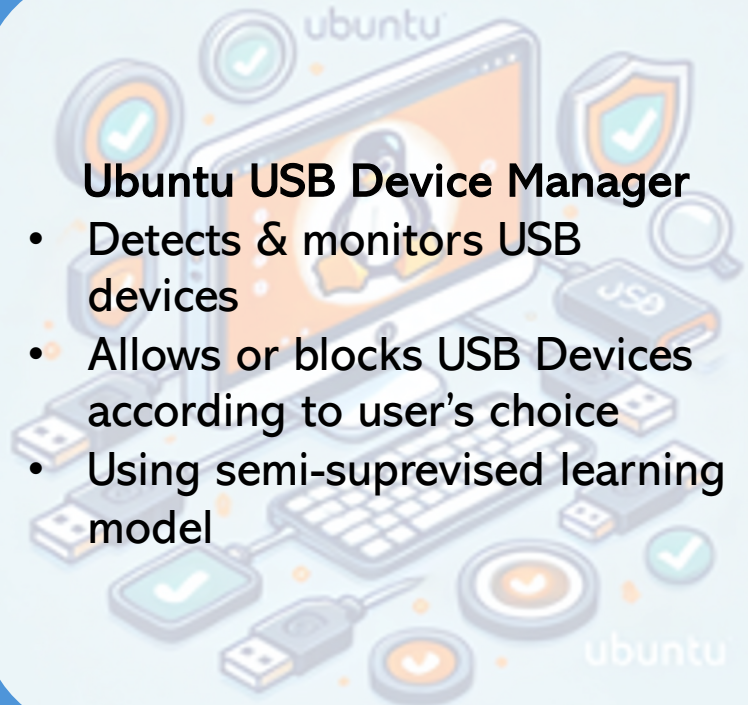
- Halel Itzhaki
- 322989674



Introduction

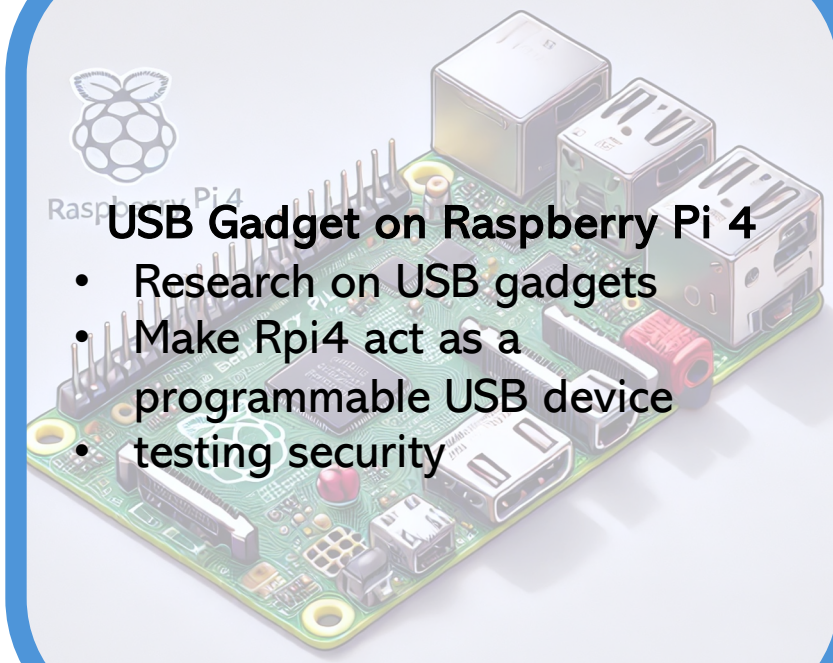
- Importance of USB devices in modern computing.
- USB devices as potential security threats.

USB Protection Project's Components



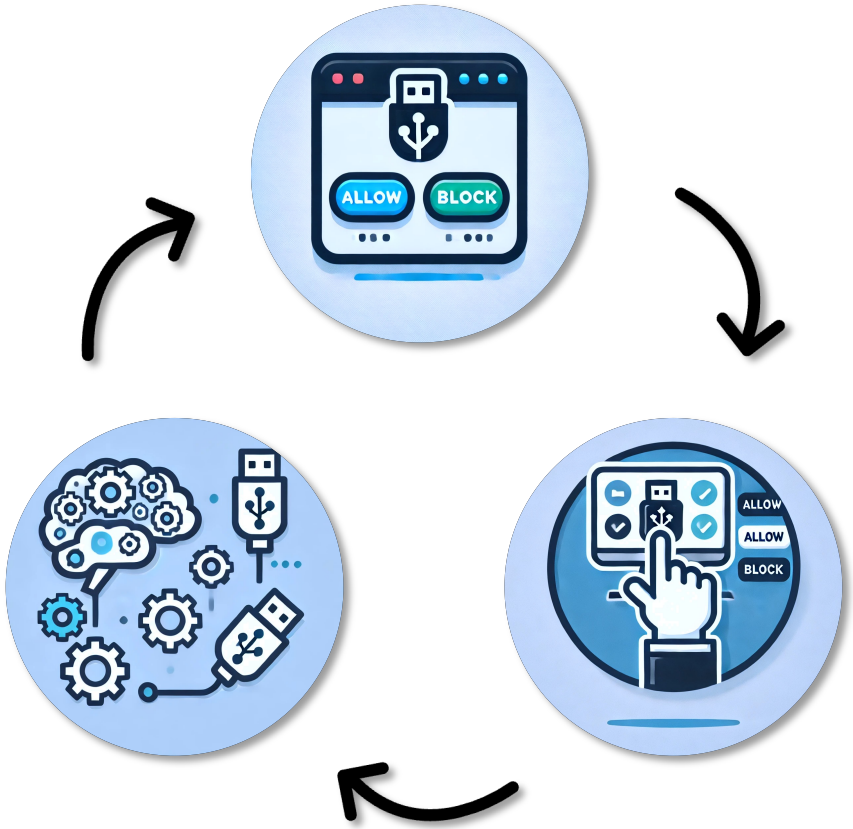
Ubuntu USB Device Manager

- Detects & monitors USB devices
- Allows or blocks USB Devices according to user's choice
- Using semi-suprevised learning model



USB Gadget on Raspberry Pi 4

- Research on USB gadgets
- Make Rpi4 act as a programmable USB device
- testing security



Ubuntu Device Manager

- **Device Detection Module**:

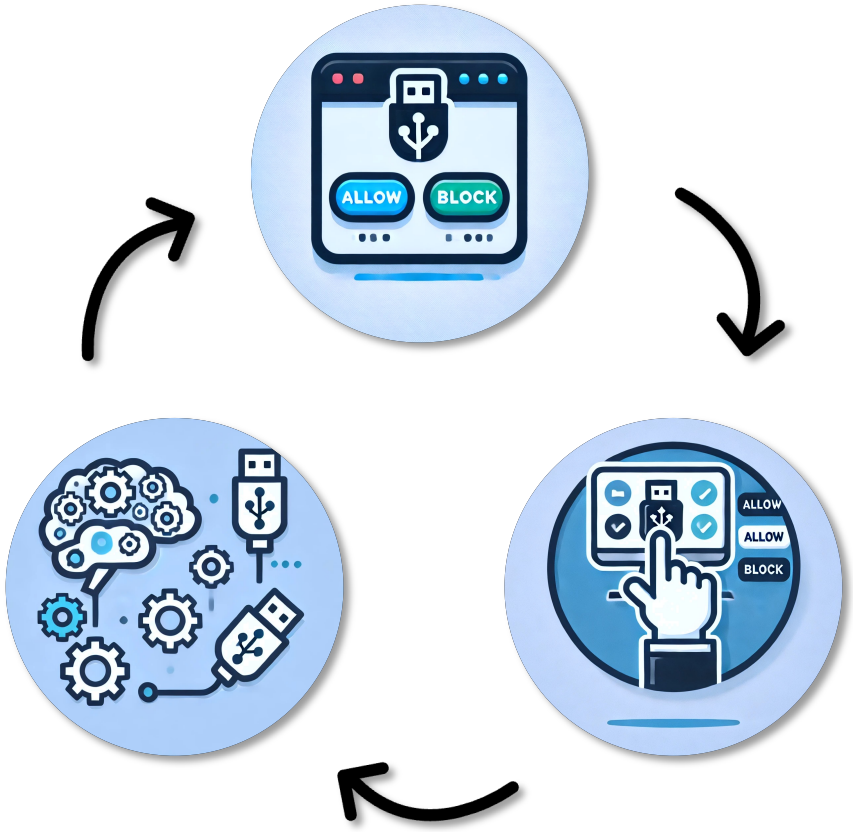
  Uses `udev` for device events.

- **User Interaction Module**:

  GUI for allowing/blocking devices.

- **Learning Module**:

  Records user actions, predicts future responses.



Ubuntu Device Manager – Related work

- **Rule-Based Systems:** USBGuard, SELinux
- **Linux udev Rules**

SELinux

USBGuard

USB Detection

```python
class USBDeviceManager:

    def monitor_usb_devices(self) -> None:
        """ Monitor USB devices and handle them based on user input and model predictions. """
        # Create interface for interacting with udev subsystem
        context = pyudev.Context()
        monitor = pyudev.Monitor.from_netlink(context)
        monitor.filter_by(subsystem='usb')  # Filtering only the USB events

        disable_auto_mount(self.root_process_launcher)

        logger.info("Monitoring USB devices. Press Ctrl+C to stop.")

        try:
            for device in iter(monitor.poll, None):
                if device.action == 'add':  # New USB device was plugged
                    # Fetch USB device information
```

USB Blocking

```python
class USBDeviceManager:

    def block_usb_device(self, device: USBDevice) -> None:
        """ Block the USB device by unmounting and powering it off. """
        logger.info(f"Blocking USB Device: {device.vendor_id} - {device.product_id}")

        block_device = get_block_device(device.device_node)

        if not block_device:
            logger.error(f"Unable to identify block device for {device.device_node}.")
            return

        # The udev rule to block the USB device
        blocking_command = f'ATTR{{idVendor}}=="{device.vendor_id}", ATTR{{idProduct}}==" \
                           f'{device.product_id}", ATTR{{authorized}}="0"'

        # Command to append the blocking rule to the udev file
        echo_command = 'tee -a /etc/udev/rules.d/99-usb-blacklist.rules'

        try:
            # Running the blocking command with root privileges
            self.root_process_launcher.execute_with_input(echo_command, blocking_command)
            update_udev_rules(self.root_process_launcher)
            logger.info(f"USB Device: {device.vendor_id} - {device.product_id} blocked successfully")

        except subprocess.CalledProcessError as e:
            logger.error(f"Failed to block the USB device: {e}")
```

Learning Model

```python
def train_model() -> None:
    for log in logs:
        if 'decision' not in log:
            continue
        vendor_id = log['vendor_id']
        vendor_ids.append(vendor_id)
        product_ids.append(log['product_id'])
        serials.append(log['serial'])
        decisions.append(1 if log['decision'] == 'allow' else 0)

        # Increment the count for each vendor being allowed
        if log['decision'] == 'allow':
            if vendor_id not in vendor_allow_count:
                vendor_allow_count[vendor_id] = 0
            vendor_allow_count[vendor_id] += 1

    # Check the balance of "allow" vs "block"
    allow_count = decisions.count(1)
    block_count = decisions.count(0)
    logger.info(f"Training data: {allow_count} allow, {block_count} block")

    # Encode features and train model as before
    le_vendor = LabelEncoder()
    le_product = LabelEncoder()
    le_serial = LabelEncoder()

    vendor_ids_encoded = le_vendor.fit_transform(vendor_ids)
    product_ids_encoded = le_product.fit_transform(product_ids)
    serials_encoded = le_serial.fit_transform(serials)

    X = np.column_stack((vendor_ids_encoded, product_ids_encoded, serials_encoded))
    y = np.array(decisions)
```

```python
def predict(device_info: {}) -> '':
    # Automatically allow only if the vendor has been allowed 5 or more times
    if vendor_allow_count.get(vendor_id, 0) >= 5:
        logger.info(f"Automatically allowing USB device from vendor {vendor_id}")
        return 'allow'

    # Handle unseen values for LabelEncoders
    try:
        vendor_encoded = le_vendor.transform([device_info['vendor_id']])[0]
    except ValueError:
        logger.warning(f"Vendor ID {device_info['vendor_id']} not recognized by encoder.")
        return None  # Prompt user for decision

    try:
        product_encoded = le_product.transform([device_info['product_id']])[0]
    except ValueError:
        logger.warning(f"Product ID {device_info['product_id']} not recognized by encoder.")
        return None  # Prompt user for decision

    try:
        serial_encoded = le_serial.transform([device_info['serial']])[0]
    except ValueError:
        logger.warning(f"Serial {device_info['serial']} not recognized by encoder.")
        return None  # Prompt user for decision

    # Convert device info to features for the model
    features = np.array([[vendor_encoded, product_encoded, serial_encoded]])

    # Model prediction (fallback if auto-allow threshold is not met)
    prediction = model.predict(features)
    return 'allow' if prediction == 1 else None
```

ByPassing USB Security

Is it possible?

ByPassing USB Security

Is it possible?

The **Flipper Zero** is a portable multi-tool for hackers and pentesters, capable of interacting with radio protocols like RFID, NFC, and infrared, making it great for security testing and hardware exploration.

ByPassing USB Security

Is it possible?

The **USB Rubber Ducky** is a keystroke injection tool disguised as a USB flash drive. It executes pre-programmed scripts rapidly, simulating keyboard input to automate attacks or tests, commonly used for penetration testing.

ByPassing USB Security

Is it possible?

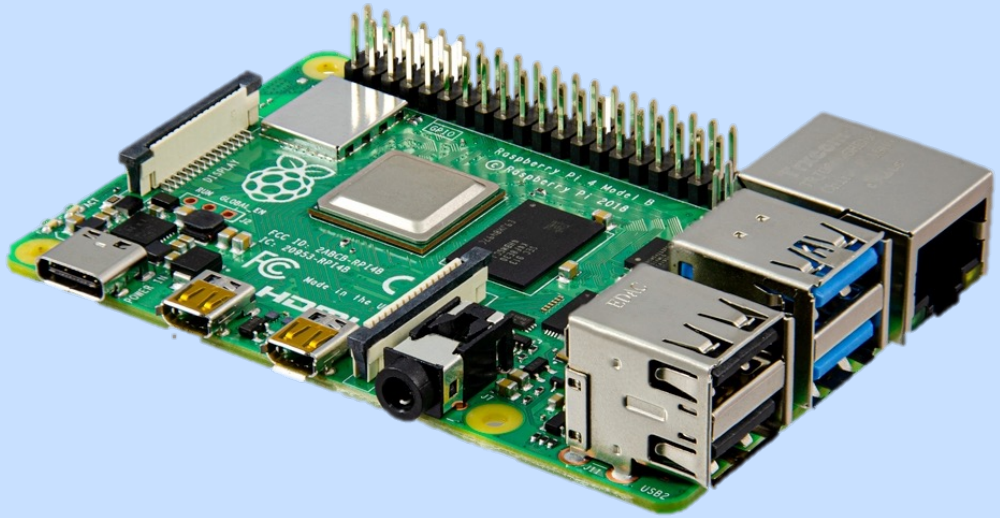
Both the **Flipper Zero** and **USB Rubber Ducky** contain a small embedded computer, which allows them to execute complex scripts and attacks independently.

Both of them are illegal in Israel

ByPassing USB Security

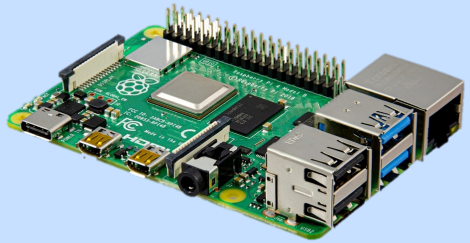
Is it possible?

The **Raspberry Pi 4** is a compact, versatile single-board computer capable of running a full Linux OS, used for education, prototyping, and various DIY projects.



ByPassing USB Security

Is it possible?

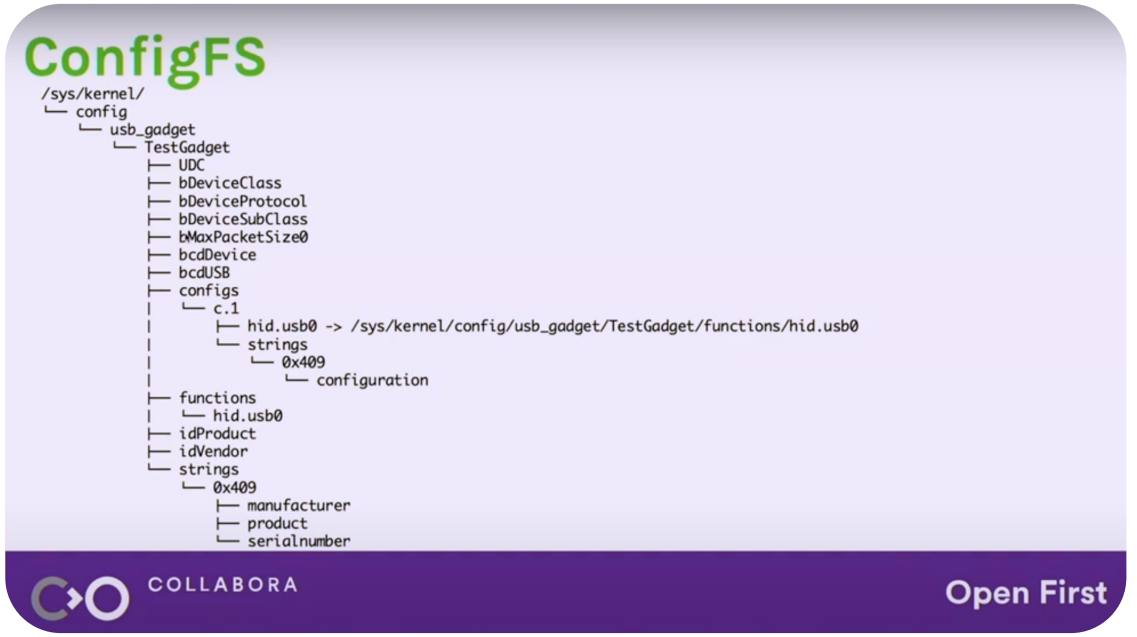


Both **Flipper Zero** and **USB Rubber Ducky**

use **`USB Gadget`** for their USB Attacks.

Research of USB Gadgets Using ConfigFS

- **ConfigFS** is a Virtual filesystem in Linux for configuring kernel objects dynamically, like USB gadgets.
- The **USB Gadget** is built from specific directories and files structure in /sys/kernel/config.
- It is possible to:
  - Configure any identifiers you want (like Vendor or Manufacturer)
  - Configure any actions you want (like keyboard or mass storage)

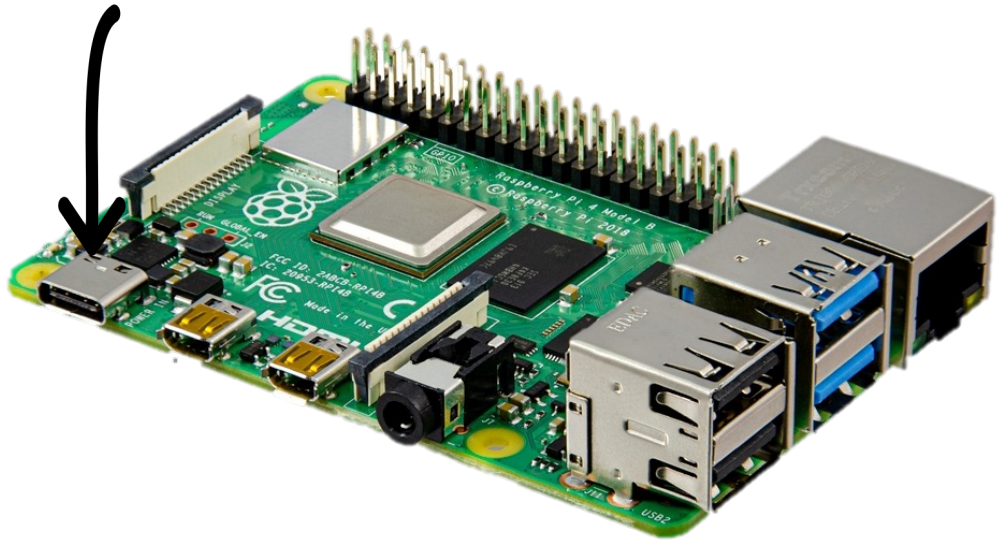


Difficulties with RPi4

- **USB Gadget** must use USB Device Controller (UDC) for running.
- The USB-C port on RPi4 is used primarily for power input, and lacked proper configuration for peripheral mode.

- Solution – use USB Hub between Host PC and RPi4

Power-In Port & UDC



Difficulties with RPi4

- **USB Gadget** must use USB Device Controller (UDC) for running.
- The USB-C port on RPi4 is used primarily for power input, and lacked proper configuration for peripheral mode.

- Solution – use USB Hub between Host PC and RPi4

Results

Ubuntu USB Device Manager

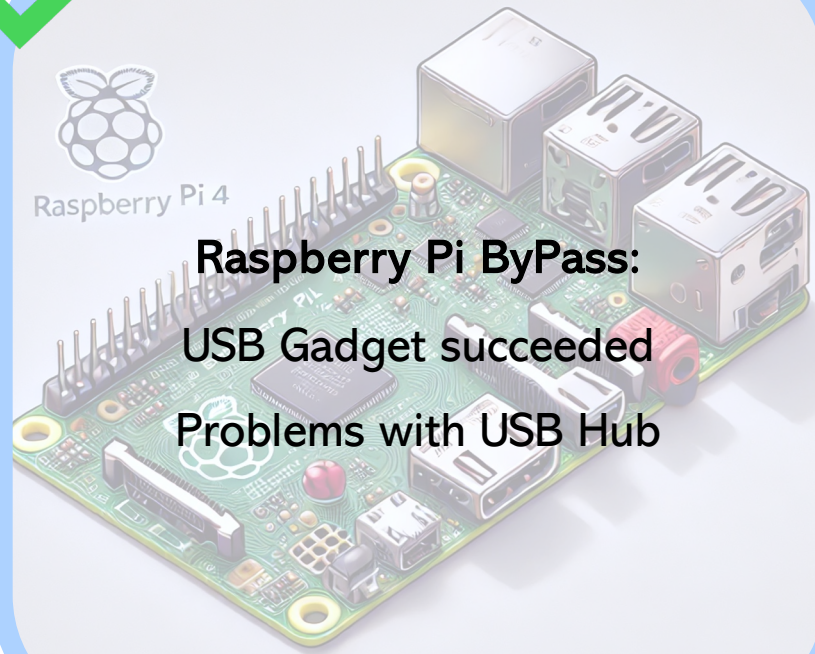
Performance:

Accurate and efficient detection.

Effectiveness of udev rules.

Learning Model:

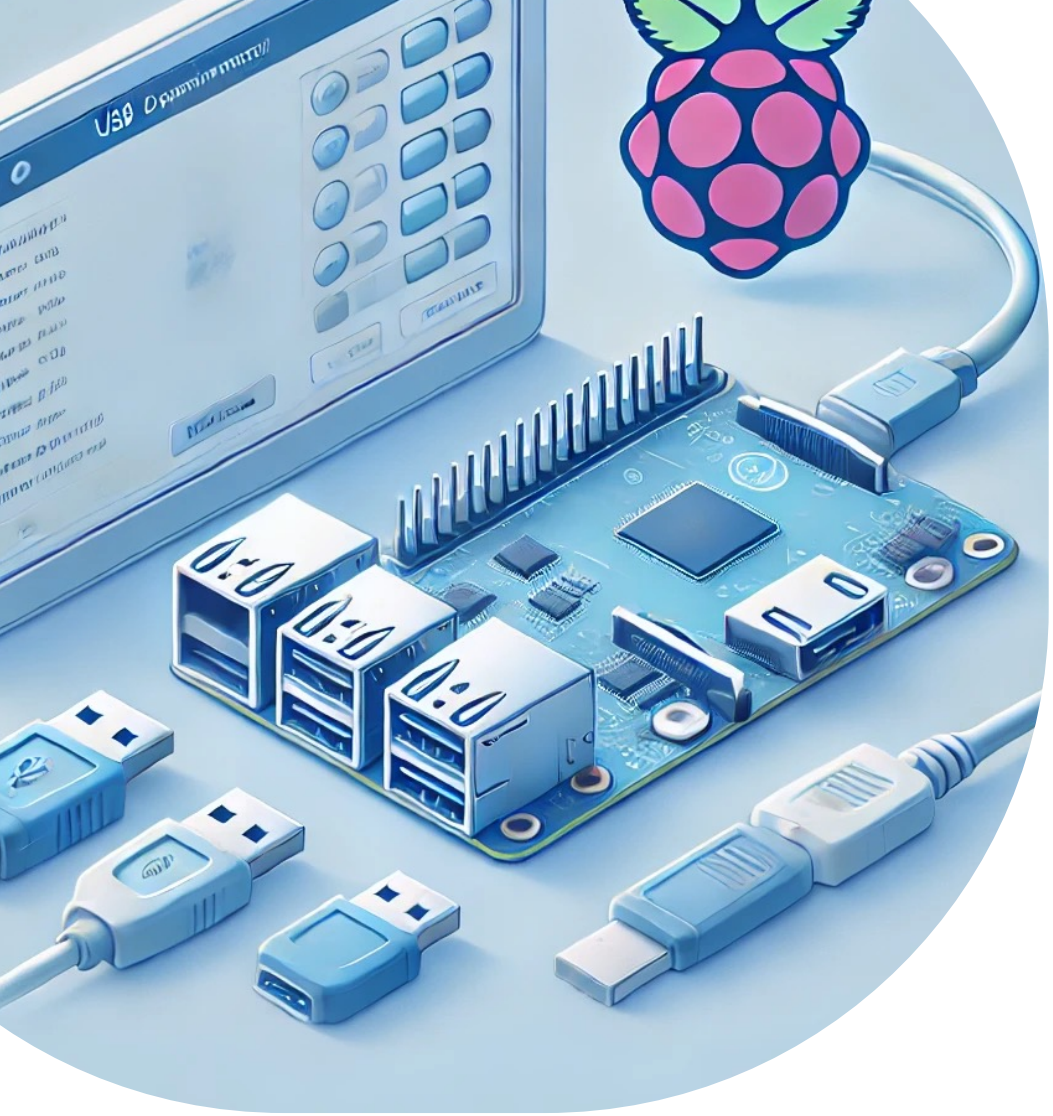
Reduced manual approvals after several interactions.



Raspberry Pi ByPass:

USB Gadget succeeded

Problems with USB Hub



Conclusion

- Dynamic USB security is crucial for evolving threats.
- Semi-supervised learning helps improve adaptability.
- Creating USB Gadget on RPi4 is possible and works.