USB Device Control Monitoring Framework Documentation Report

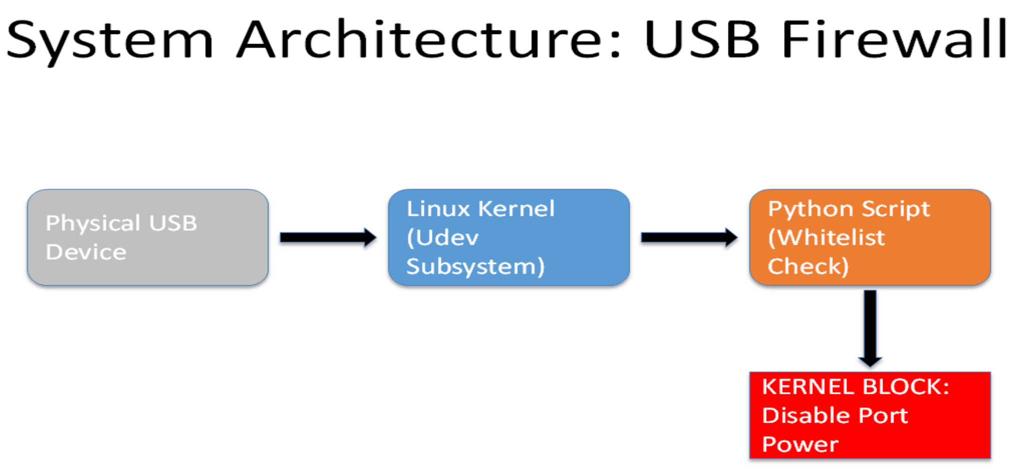
Author: Raj Bharti

1. Executive Summary

- **Objective:** To implement a host-based security control that detects and restricts unauthorized USB storage devices to prevent data exfiltration and malware introduction.
- **Scope:** Endpoint monitoring, real-time alerting, and automated blocking logic.

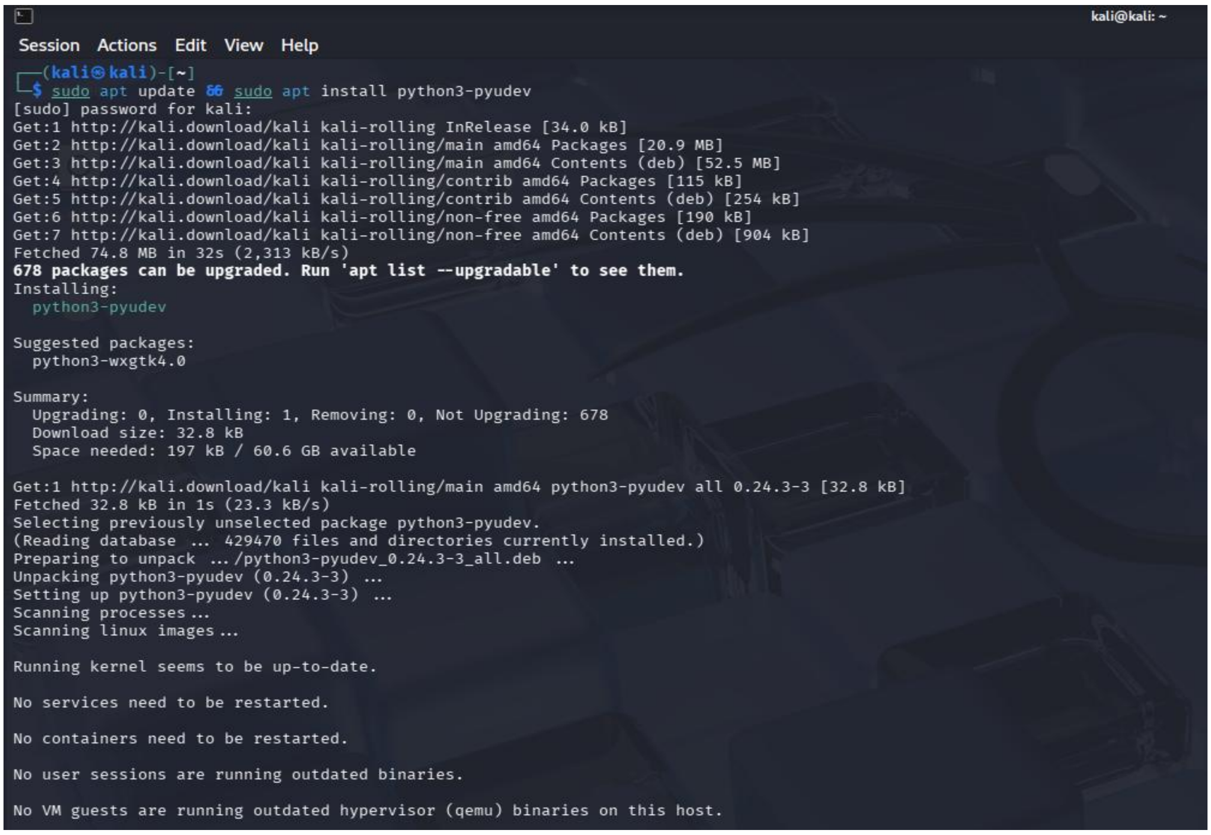
2. Technical Implementation

- **Language:** Python
- **Key Libraries:** pyudev (Linux Device Management), os, time.
- **Environment:** Kali Linux.
- The system is designed as a user-space driver that interacts with the Linux Kernel's **udev** subsystem. It listens for hardware events in real-time and enforces a strict "Default Deny" policy.


System Architecture: USB Firewall
Physical USB Device
Linux Kernel (Udev Subsystem)
Python Script (Whitelist Check)
KERNEL BLOCK: Disable Port Power

3. Implementation Steps & Evidence

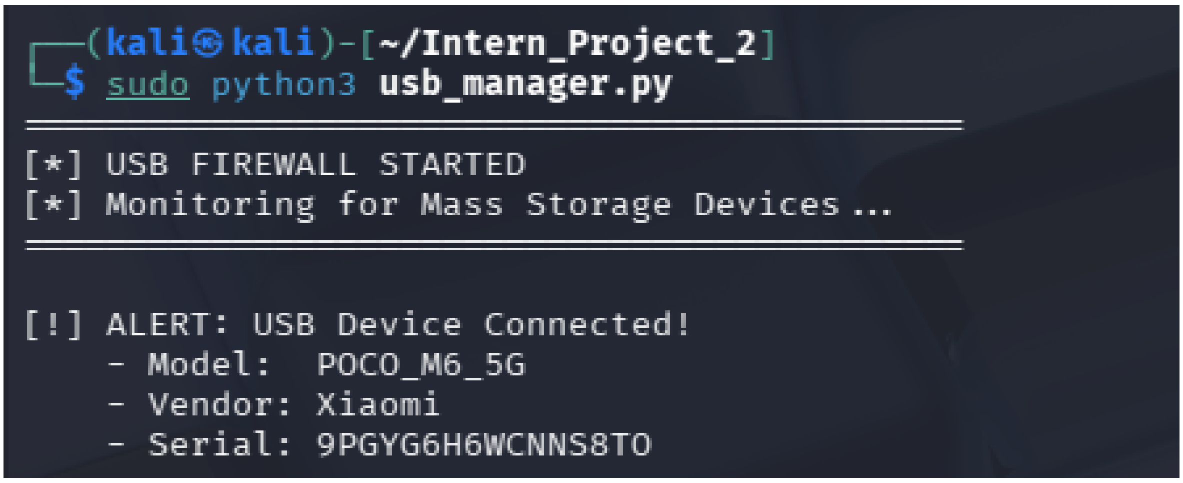
3.1. Environment Setup & Tool Installation



```
┌──(kali㉿kali)-[~]
└─$ sudo apt update && sudo apt install python3-pyudev
[sudo] password for kali:
Get:1 http://kali.download/kali kali-rolling InRelease [34.0 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.9 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.5 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [115 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [254 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [190 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [904 kB]
Fetched 74.8 MB in 32s (2,313 kB/s)
678 packages can be upgraded. Run 'apt list --upgradable' to see them.
Installing:
  python3-pyudev

Suggested packages:
  python3-wxgtk4.0

Summary:
  Upgrading: 0, Installing: 1, Removing: 0, Not Upgrading: 678
  Download size: 32.8 kB
  Space needed: 197 kB / 60.6 GB available

Get:1 http://kali.download/kali kali-rolling/main amd64 python3-pyudev all 0.24.3-3 [32.8 kB]
Fetched 32.8 kB in 1s (23.3 kB/s)
Selecting previously unselected package python3-pyudev.
(Reading database ... 429470 files and directories currently installed.)
Preparing to unpack .../python3-pyudev_0.24.3-3_all.deb ...
Unpacking python3-pyudev (0.24.3-3) ...
Setting up python3-pyudev (0.24.3-3) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

3.2. Detection Logic (The "Listener")

The script initializes a netlink socket monitor to capture kernel signals. Upon detecting an **add** event from the USB subsystem, it extracts the device's Vendor ID, Model, and Serial Number.

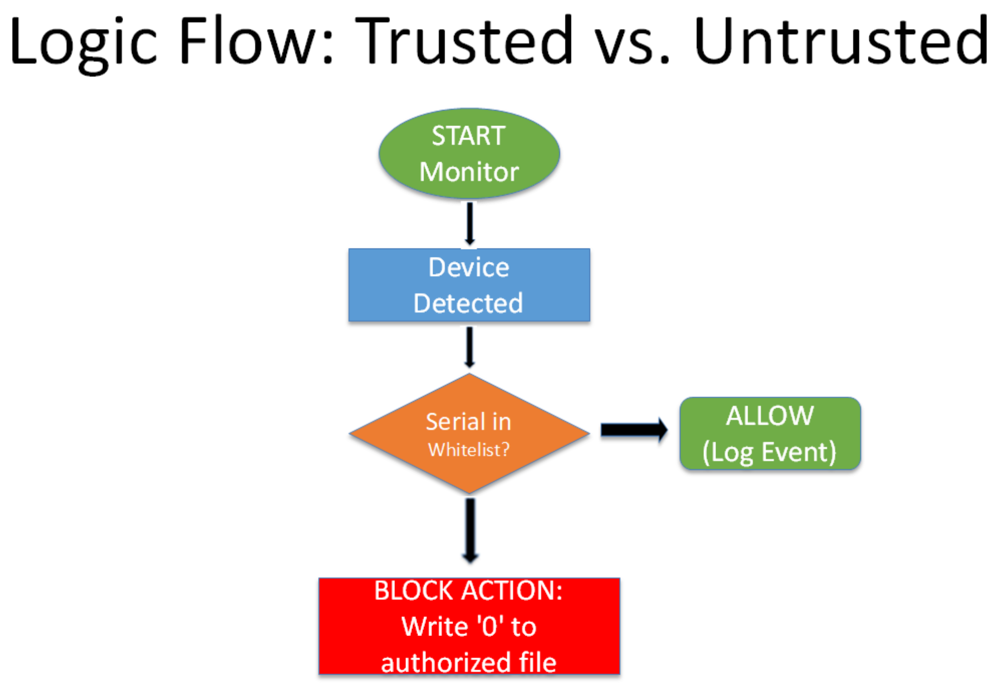


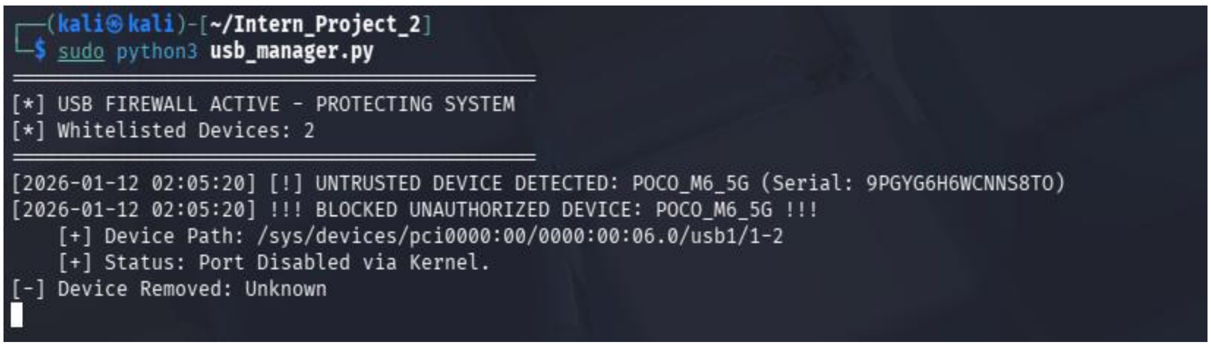
```
┌──(kali㉿kali)-[~/Intern_Project_2]
└─$ sudo python3 usb_manager.py
════════════════════════════════════════════════
[*] USB FIREWALL STARTED
[*] Monitoring for Mass Storage Devices...
════════════════════════════════════════════════

[!] ALERT: USB Device Connected!
    - Model:  POCO_M6_5G
    - Vendor: Xiaomi
    - Serial: 9PGYG6H6WCNNS8TO
```

3.3. Blocking Logic (The "Firewall")

The core security control involves comparing the device's Serial Number against a pre-defined **USB_WHITELIST**.

- **Trusted:** If the serial matches, the device is logged and allowed to mount.
- **Untrusted:** If no match is found, the system immediately writes **0** to the device's **/sys/bus/usb/.../authorized** file, electrically disabling the data port.

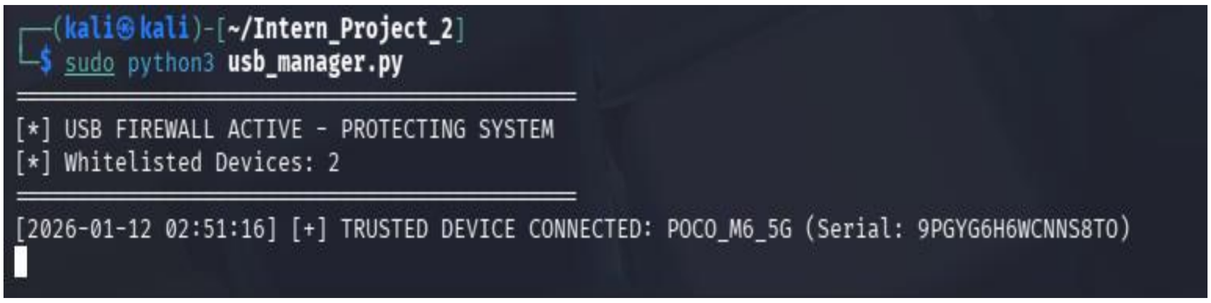

Logic Flow: Trusted vs. Untrusted
START Monitor
Device Detected
Serial in Whitelist?
ALLOW (Log Event)
BLOCK ACTION: Write '0' to authorized file



```
┌──(kali㉿kali)-[~/Intern_Project_2]
└─$ sudo python3 usb_manager.py
==================================================
[*] USB FIREWALL ACTIVE - PROTECTING SYSTEM
[*] Whitelisted Devices: 2
==================================================
[2026-01-12 02:05:20] [!] UNTRUSTED DEVICE DETECTED: POCO_M6_5G (Serial: 9PGYG6H6WCNNS8TO)
[2026-01-12 02:05:20] !!! BLOCKED UNAUTHORIZED DEVICE: POCO_M6_5G !!!
    [+] Device Path: /sys/devices/pci0000:00/0000:00:06.0/usb1/1-2
    [+] Status: Port Disabled via Kernel.
[-] Device Removed: Unknown
```

Section 3.4: Whitelist Verification

To verify operational usability, a known device (Serial ending in ...8TO) was added to the whitelist. The system successfully validated the serial number and permitted the connection.



```
┌──(kali㉿kali)-[~/Intern_Project_2]
└─$ sudo python3 usb_manager.py
══════════════════════════════════════════════════
[*] USB FIREWALL ACTIVE - PROTECTING SYSTEM
[*] Whitelisted Devices: 2
══════════════════════════════════════════════════
[2026-01-12 02:51:16] [+] TRUSTED DEVICE CONNECTED: POCO_M6_5G (Serial: 9PGYG6H6WCNNS8TO)
```

Section 4: Conclusion

This project successfully demonstrated a host-based "USB Firewall" capable of preventing unauthorized peripherals from interacting with the operating system. By leveraging kernel-level authorization controls, the system mitigates risks associated with BadUSB attacks and physical data exfiltration.

Appendix A: Project Source Code

```python
import pyudev
import time
import sys
import os

# ================================================================================
# PROJECT 2: USB DEVICE CONTROL TOOLKIT
# AUTHOR: [Your Name]
# PURPOSE: Detects USB Mass Storage devices and enforces a Whitelist policy.
# ================================================================================

# --- CONFIGURATION ---
# REPLACE with your actual device Serial Number found during testing.
USB_WHITELIST = ["9PGYG6H6WCNNS8TO", "ANOTHER_TRUSTED_SERIAL"]

def log_event(message):
    """
    Logs events to both the terminal and a local log file for auditing.
    """
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
    formatted_msg = f"[{timestamp}] {message}"

    print(formatted_msg)

    # Append to log file (Deliverable 3 Evidence)
    with open("usb_security.log", "a") as f:
        f.write(formatted_msg + "\n")

def block_device(device):
    """
    Disables the USB device by writing '0' to the kernel 'authorized' interface.
    This effectively cuts power/data to the specific USB port.
    """
    try:
        # The 'authorized' file controls if the OS allows the device to communicate
        auth_path = os.path.join(device.sys_path, 'authorized')

        if os.path.exists(auth_path):
            with open(auth_path, 'w') as f:
                f.write('0') # 0 = Disabled, 1 = Enabled

            log_event(f"!!! ACTION TAKEN: BLOCKED UNAUTHORIZED DEVICE: 
{device.get('ID_MODEL')} !!!")
            print(f"    [+] Path: {device.sys_path}")
            print("    [+] Status: Port Disabled via Kernel.")
        else:
            log_event(f"[-] Warning: Could not find 'authorized' control file for 
{device.get('ID_MODEL')}")

    except Exception as e:
        log_event(f"[!] Error attempting to block: {str(e)}")

def monitor_usb():
    """
    Main loop that listens for Kernel Udev events.
    """
    context = pyudev.Context()
    monitor = pyudev.Monitor.from_netlink(context)

    # Filter only for 'usb' subsystem events
    monitor.filter_by(subsystem='usb')

    print("=============================================")
    print("[*] USB FIREWALL ACTIVE - PROTECTING SYSTEM")
    print(f"[*] Whitelisted Devices: {len(USB_WHITELIST)}")
    print("=============================================")

    # Continuous listening loop
    for device in iter(monitor.poll, None):
        if device.action == 'add':
            # Wait briefly for attributes to initialize
            time.sleep(1)

            # --- INPUT SANITIZATION ---
            # Get the raw serial and remove any hidden non-alphanumeric characters
            # This fixes issues with hidden null bytes or spacing
            raw_serial = device.get('ID_SERIAL_SHORT', 'Unknown')
            serial = ''.join(char for char in str(raw_serial) if char.isalnum())
            model = device.get('ID_MODEL', 'Unknown')

            # Ignore internal USB hubs/interfaces, focus on the device itself
            if device.device_type == 'usb_interface':
                continue

            # --- DECISION ENGINE ---
            if serial in USB_WHITELIST:
                log_event(f"[+] ALLOWED: Trusted Device Connected - {model} (Serial: {serial})")
            else:
                log_event(f"[!] ALERT: Untrusted Device Detected - {model} (Serial: {serial})")
                block_device(device)

        elif device.action == 'remove':
             model = device.get('ID_MODEL', 'Unknown')
             print(f"[-] Device Removed: {model}")

if __name__ == "__main__":
    # Script must be run as Root to access /sys/ files for blocking
    if os.geteuid() != 0:
        print("[-] ERROR: This script requires ROOT privileges.")
        print("    Run with: sudo python3 usb_manager.py")
        sys.exit(1)

    try:
        monitor_usb()
    except KeyboardInterrupt:
        print("\n[*] Stopping USB Firewall...")
        sys.exit()
```