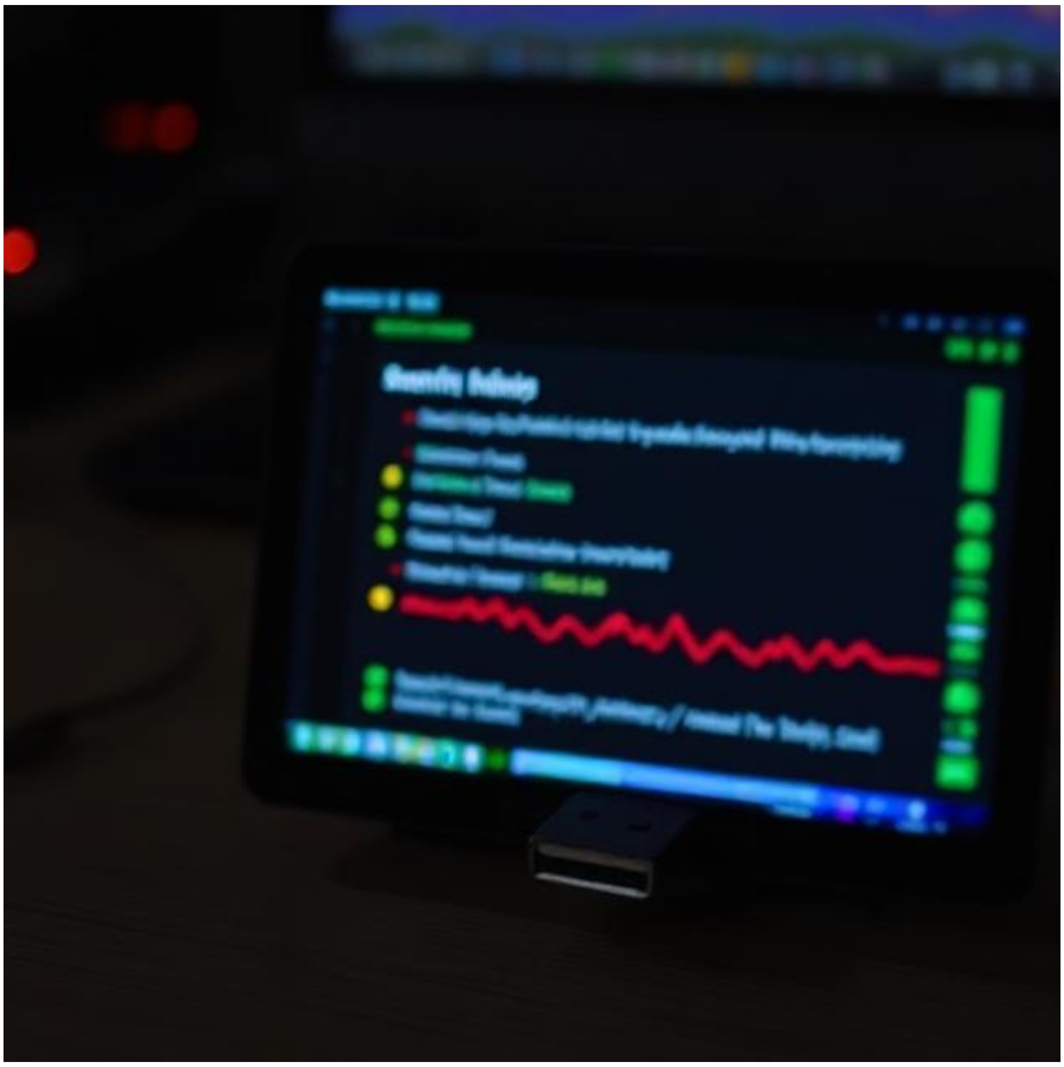
Detecting Malicious Communication in Air-Gap Systems
-
USB Monitoring Design

Kelly Zhang



Inhoud

Version

Version	Name	Changes
1.0	Kelly Zhang	Everything : context, design(libraries & commands, UML diagram) and reference

Context

Air-gap systems, designed to be isolated from external networks, are widely regarded as one of the most secure configurations for protecting sensitive data and critical infrastructure. Physical separation from the internet or other networks significantly reduces the risk of direct cyberattacks. However, this isolation does not guarantee absolute security, as sophisticated adversaries have increasingly found ways to exploit vulnerabilities through indirect communication channels. Malicious communication, often hidden or disguised within normal system operations, poses a growing threat to air-gap systems, making traditional security measures insufficient.

To protect air-gapped systems, we made an analysis document where we compared hardware-based and software-based detection systems. From that, we found out that both systems play an important role in protecting an air-gapped setup. In the document, we also gave several recommendations to build a software-based malware detection system for air-gapped environments — without depending on traditional cloud-based antivirus solutions.

One such recommendation is: The system should detect data being written to a USB from the air-gapped machine, because malware present on an air-gapped system can replicate itself and extract data using USB flash drives to collect sensitive information such as credentials, screenshots, and more. This type of attack is often the only method of exfiltration and infection.

That’s why this document includes a UML design based on this recommendation, so that during implementation you don’t get stuck or confused. Here, I explain what the design includes and why I made certain choices.

Design

Here, I explain what the UML diagram includes and why I made certain choices.

Libraries & commands

Notify VS fanotify VS usbmon

Notify is a high-level Linux API for monitoring individual files or directories. It allows you to easily capture events like *"file opened"*, *"deleted"*, or *"modified"*. It also works in real-time and doesn't require root privileges.[3]

Usbmon is a Linux kernel interface used for logging low-level USB traffic between the kernel and USB devices. It shows actual USB packets, including descriptors, bulk transfers, control transfers, etc. Since it captures a lot of raw data, you need to do heavy filtering, and it's not meant for real-time file-event monitoring[2]. It's more suitable for USB forensics and analysis.

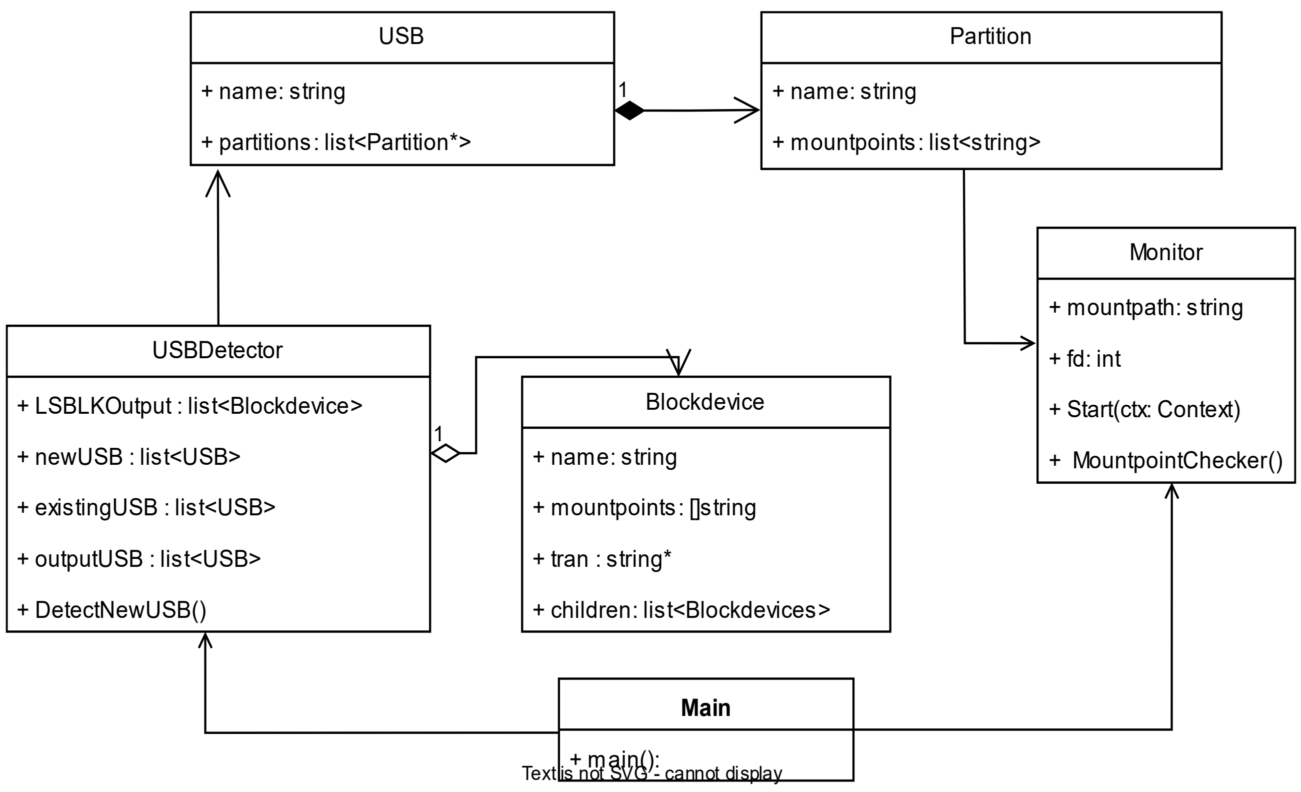
Fanotify, on the other hand, is quite similar to notify – both are Linux APIs and can be used in real time. However, fanotify can monitor entire USB sticks or mount points, not just individual files[1]. It operates at a lower level, allowing it to detect more detailed events, such as attribute changes, file movements, and more. Real-time monitoring is crucial here to prevent further exfiltration of data[1].

Lsblk VS udev

Udev is the Linux device manager for dynamically creating, removing, and managing device nodes when hardware is added or removed[5]. It uses rules to recognize devices and trigger actions (e.g., loading drivers). While udev is dynamic and customizable, it lacks the ability to find mountpoints, and for airgap systems, downloading additional packages is a hassle. Worse, those packages would need to be transferred via the very USB stick we're trying to avoid, creating extra dependencies[5].

This is where lsblk comes in. It's a command-line tool included in most modern Linux distros that lists block devices with their properties—like mountpoints, storage size, filesystem, and more[4]. You get a quick, clean overview in a tree structure, but it's read-only, you can't modify devices with it. By running lsblk repeatedly and comparing old vs. new block devices, you can detect when a USB is added or removed. This approach keeps dependencies to a minimum, which is exactly what we want.

UML diagram



USBDetector

The USBDetector detects when ether there are new USB-devices plugged in the computer by comparing new results from the lsblk and “old” data we already have.

Attribute:

- Blockdevices []BlockDevice: are result from the lsblk-command.
- NewUSB, OutputUSB, ExistingUSB []USB: are used for comparing and chenking for new USBs connected.

Importance in UML:

- This class controls the entire detection process.
- In the diagram, we see two arrows coming from USBDetector pointing to USB and BlockDevice, because it directly processes these structures.

USB

Responsibility: Represents a detected USB device.

Attributes:

- Name string: the name of the device (e.g., sdb)

- Partitions []*Partition: a list of partitions on this device (e.g., sdb1, sdb2, etc.)

Relationships:

- Composition (filled diamond) with Partition:
  → This means a USB can contain multiple partitions, but a Partition cannot exist on its own (independent of a USB). When you unplug the USB, its partitions no longer exist either.

Partition

Responsibility: Represents a single partition of a USB stick.

Attributes:

- Name string: name of the partition, like sdb1, sdb2, sda1, etc.
- Mountpoints []string: mount locations where this partition is accessible, like /mnt/usb/disk, etc.

Note :

- It's needed to determine which mountpoints should be monitored.
- It's used by the Monitor.

Monitor

Responsibility: Tracks changes in file events within a mount path.

Attributes:

- Mountpath string: the path where the USB is mounted and where monitoring happens.
- fd C.int: the Fanotify descriptor.

Relationship:

- Depends on Partition (→ directed association): because it uses Partition.Mountpoints to start a Monitor.

BlockDevice

Responsibility: Structure returned by lsblk (via JSON).

Attributes:

- Name string

- Mountpoints []string
- Tran *string (used to identify 'usb' type)
- Children []BlockDevice

Note:

- The UML diagram also shows the recursive relationship (→ Children is a list of BlockDevice)
  ⇒ This is a self-referential aggregation.

References

[1] fanotify_mark(2) - Linux manual page. (n.d.). https://www.man7.org/linux/man-pages/man2/fanotify_mark.2.html

[2] *usbmon — The Linux Kernel documentation*. (n.d.). https://www.kernel.org/doc/html/v6.15-rc7/usb/usbmon.html

[3] *inotify(7) - Linux manual page*. (n.d.). https://www.man7.org/linux/man-pages/man7/inotify.7.html

[4] GeeksforGeeks. (2024, September 15). *How to List All Block Devices in Linux | lsblk Command*. GeeksforGeeks. https://www.geeksforgeeks.org/lsblk-command-in-linux-with-examples/

[5] udev(7) - Linux manual page. (n.d.). https://www.man7.org/linux/man-pages/man7/udev.7.html