Gatekeeper Toolkit - Project Documentation

(**Gatekeeper** : USB Device Control & Monitoring Framework **)**

Author: Om Fulsundar
Version: 1.0

Contents :

- Abstract
- Keywords
    1. Introduction and Motivation
    2. Design and Architecture
    3. Implementation Details
    4. Workflow and Diagrams
    5. Installation
    6. Usage Overview
    7. Reporting and Outputs
    8. Testing and Validation
    9. Results and Observations
    10. Limitations
    11. Future Work
    12. Conclusion
- References

Abstract :

The Gatekeeper is a lightweight and modular framework in Python that offers monitoring, policy enforcement, and auditing of USB mass-storage devices. It identifies hardware events, identifies device types based on vendor and product IDs and corresponding serial numbers, enforces decisions based on a blocklist/allowlist approach, audits file movements via SHA-256 hashing algorithms, and offers a daily consolidated report for use in an academic setting and in responding to incidents. The tool is developed in a way that makes it easy to modify and extend individual components. The overall structure is divided based on the needs and requirements of individual modules to help those who are involved in academia and/or research modify and extend the overall functionality without necessarily rebuilding the entire program. The Gatekeeper focuses on reproducibility and creating transparent audit trails to aid in a thorough evaluation of a system after the event.

Keywords :

USB monitoring, allowlist, blocklist, file audit, SHA256, pyudev, watchdog, endpoint security, log aggregation, report generation.

---

1. Introduction and Motivation :

Removable media is also an ever-popular method for malware distribution and data theft. Gatekeeper was conceived as a means of creating a tangible example of defensive strategies for protecting against exploits targeting USB-based media. The project has two target audiences: security professionals looking to reproduce a proof of concept codebase, and students/researchers looking to deconstruct the codebase as a means of learning about device monitoring. It is not necessarily focused on remedying the risks, but rather on the audit and verification aspects of a system.

---

2. Design and Architecture :

Design Principles :

- **Modularity** — Each capability is implemented as an independent module to simplify testing and extension.
- **Transparency** — Console output mirrors saved logs so runtime behavior is easy to verify.
- **Reproducibility** — All events and audit entries are persisted to logs and aggregated into daily reports.
- **Extensibility** — New policy rules, notification channels, or platform backends can be added with minimal changes.

Components :

- **main.py** — Orchestrator that initializes modules, manages the event loop, and handles graceful shutdown.
- **monitor.py** — Subscribes to udev events and enumerates baseline devices at startup.
- **identify.py** — Extracts and normalizes device attributes for fingerprinting.
- **policy.py** — Implements allowlist/blocklist logic and returns ALLOW or DENY decisions.
- **alert.py** — Records policy decisions and prints concise notifications.
- **audit.py** — Observes mounted device paths, records file events, and computes SHA256 checksums.
- **report.py** — Aggregates logs into daily timestamped reports.

---

3. Implementation Details :

Event Monitoring :

Gatekeeper uses pyudev to detect USB insertion and removal events in real time. At startup the monitor enumerates connected devices to establish a baseline and avoid false positives for preexisting mounts.

Device Identification :

The identify module extracts vendor ID, product ID, and serial number from event metadata. Values are normalized to a consistent format for reliable allowlist comparisons.

Policy Enforcement :

The policy module evaluates device fingerprints against stored allowlist and blocklist entries. Decisions are binary: **ALLOW** triggers auditing, **DENY** triggers immediate logging and halts further processing for that device.

Auditing :

When a device is allowed and mounts, the audit module uses watchdog to observe file creation, modification, and deletion events on the mount path. For transferred files the module computes SHA256 hashes and records entries with timestamps and file metadata.

Reporting :

The report module aggregates monitor, alert, and audit logs into a daily report named gatekeeper_report_YYYY-MM-DD.txt. Reports are saved in data/logs/ to keep historical records separate and manageable.

Graceful Shutdown :

main.py handles KeyboardInterrupt to stop observers and save the final report cleanly, avoiding stack traces and ensuring log integrity.

---

4. Workflow and Diagrams :

Execution Workflow

1. **System Initialization** — enumerate baseline devices and prepare logging.
2. **Event Detection** — monitor detects USB insertion or removal.
3. **Identification** — extract vendor ID, product ID, and serial number.
4. **Policy Evaluation** — decision diamond: ALLOW or DENY.
    - **DENY** — log alert and stop processing for that device.
    - **ALLOW** — record decision, wait for mount path, and start audit.
5. **Auditing** — capture file events and compute SHA256 hashes for transferred files.
6. **Reporting** — aggregate logs into a daily report.
7. **Continue Monitoring** — return to event detection for subsequent devices.

Diagrams :

Flowchart and workflow diagrams were created to illustrate the decision branching at policy evaluation and the audit/reporting path for allowed devices. These diagrams are available in document section of GitHub repository of project.

---

5. Installation :

Downloadable Libraries

Gatekeeper relies primarily on Python's standard library. The following third-party libraries are required and can be installed via pip:

- **pyudev** — udev event monitoring on Linux.
- **watchdog** — filesystem event monitoring for audits.

Install packages with:

```
pip install pyudev watchdog
```

Note: For Windows deployments, equivalent libraries or APIs (for example, pywin32 and wmi) are required; the current implementation targets Linux.

---

6. Usage Overview :

Starting Gatekeeper

Run the orchestrator:

```
python3 main.py
```

Typical Operation

- The tool enumerates baseline devices at startup.
- When a USB device is connected, Gatekeeper detects the event and extracts device attributes.

- The policy module returns **ALLOW** or **DENY**.
    - **ALLOW**: Gatekeeper waits for the device to mount, starts auditing, and logs file transfers.
    - **DENY**: Gatekeeper logs the alert and prevents further processing for that device.
- Daily reports are saved to data/logs/.

Operational Notes

- Mount detection polls common mount roots (/media, /run/media, /mnt, /media/<username>) with a configurable timeout.
- Administrators should maintain allowlist/blocklist entries for accurate policy decisions.
- For Windows support, replace the monitor backend with WMI/Win32 equivalents.

---

7. Reporting and Outputs :

Gatekeeper produces the following artifacts:

- **Monitor log** (usb_monitor.log) — records device events and timestamps.
- **Alert log** (usb_alert.log) — records policy decisions and rationale.
- **Audit log** (usb_audit.log) — records file events and SHA256 checksums.
- **Daily report** (gatekeeper_report_YYYY-MM-DD.txt) — aggregated summary of the above logs.

Reports are human-readable and suitable for archival or inclusion in incident reports.

---

8. Testing and Validation :

Test cases executed during development:

- **Idle test** — run Gatekeeper with no devices connected to confirm no false positives.
- **Allowlist test** — connect a registered device and verify ALLOW, mount detection, and audit entries.
- **Block test** — connect an unregistered device and verify DENY and alert logging.

- **File transfer test** — copy files to/from an allowed device and confirm SHA256 entries in audit logs.
- **Removal test** — unplug device and confirm audit termination and report generation.

Screenshots and log excerpts demonstrating each test case are included in the repository's screenshots file.

---

9. Results and Observations :

- Gatekeeper reliably detected udev events and extracted device attributes in test environments.
- Policy enforcement correctly blocked unregistered devices and allowed registered ones.
- The audit module captured file events and produced SHA256 hashes for transferred files, enabling integrity verification.
- Daily report generation produced concise summaries that were easy to review and archive.
- Graceful shutdown prevented noisy stack traces and ensured observers were stopped cleanly.

---

10. Limitations :

- **Platform specificity** — current implementation is Linux-centric; Windows requires alternate backends.
- **Mount detection** — polling common mount points may miss nonstandard automount configurations.
- **Spoofing risk** — device attribute spoofing is possible; hardware attestation is outside the current scope.
- **Resource considerations** — continuous filesystem observers may increase resource usage on heavily loaded systems.
- **Scope of audit** — Gatekeeper records file events and hashes but does not perform content scanning or malware analysis.

---

11. Future Work :

- Externalize configuration to a config.json for paths, timeouts, and policy settings.
- Implement log rotation and retention policies (e.g., keep last N reports, compress older logs).
- Add a Windows backend using WMI/Win32 APIs (pywin32, wmi).
- Improve audit observer lifecycle to ensure observers stop immediately on device removal.
- Add structured report exports (CSV/JSON) and optional SIEM integration.
- Integrate device reputation or threat intelligence feeds to enhance policy decisions.

---

12. Conclusion :

Gatekeeper reveals the practical approach to the control of USB devices, supporting the audit process. By using real time hardware monitoring, policy enforcement, and file auditing methodologies with SHA-256 hashing, artefacts are removed, leaving a clear path for incident investigations and policy validation. The modularity of this unique tool makes it easy to enhance, where new detection methods can be added, notifications can be implemented, or a new back-end platform can be implemented easily.

Gatekeeper offers some immediate value for use within an academic and small-scale operation as a tool for endpoint security through reduced suspected removable media activity, verifiable log files for post-event analysis, and as a teaching tool for endpoint security principles. As Gatekeeper continues to add features such as externalized configuration methods, support for the Windows operating system, exports of structured data, and centralized logging, the tool can progress as a more solid endpoint security utility.

References :

- pyudev documentation — udev event handling on Linux.
- watchdog documentation — filesystem event monitoring.
- Python standard library documentation (os, hashlib, datetime, sys).
- Endpoint security literature on removable media risks and mitigation strategies.