Software Architecture and Design Specification

**Project:** System Performance Tracker

**Version:** 1.0

**Authors:** Karan M, K Adithya Shetty, Pavan Kumar G, Purandar Balasa

**Date:** 02-10-2025

**Status:** Draft

# Revision History

Version | Date | Author | Change Summary

# Approvals

Role | Name | Signature/Date

**1. Introduction**

**1.1 Purpose**

This document specifies the architecture and design of the System Performance Tracker. It details the system's components, architectural patterns, design decisions, and technology stack required to meet the specified requirements.

**1.2 Scope**

The scope covers the architecture for monitoring core system performance metrics on a Linux/Unix system, including CPU usage, memory utilization, disk I/O, network statistics, and process management. It includes designs for both standard and privileged user access modes.

**1.3 Audience**

This document is intended for Developers, QA Engineers, System Integrators, Maintenance Technicians, and Assessment Evaluators.

**1.4 Definitions**

* **CPU:** Central Processing Unit
* **PID:** Process Identifier
* **TUI:** Terminal User Interface
* **CLI:** Command Line Interface
* **RBAC:** Role-Based Access Control

**2. Document Overview**

**2.1 How to use this document**

This document provides the architectural blueprint for the System Performance Tracker. It should be used by developers to guide implementation and by QA engineers to understand the system's internal structure for testing purposes.

**2.2 Related Documents**

* System Performance Tracker SRS v1.0
* System Performance Tracker Test Plan v1.0
* Requirements Traceability Matrix (RTM)

**3. Architecture**

**3.1 Goals & Constraints**

* **Goals:**
  + **Performance:** The system must provide near real-time updates (≤1s latency) with low overhead (≤ 2% CPU usage).
  + **Reliability:** The system must be stable, handle rapid process changes without crashing, and recover from transient faults within 5 seconds.
  + **Security:** The system must enforce a strict privilege model, ensuring sensitive metrics are only accessible to administrative users.
  + **Usability:** The TUI must be clear, readable, and support accessibility features.
* **Constraints:**
  + The user interface is constrained to a Terminal User Interface (TUI) rather than a graphical one.
  + The application has a direct dependency on the Linux/Unix

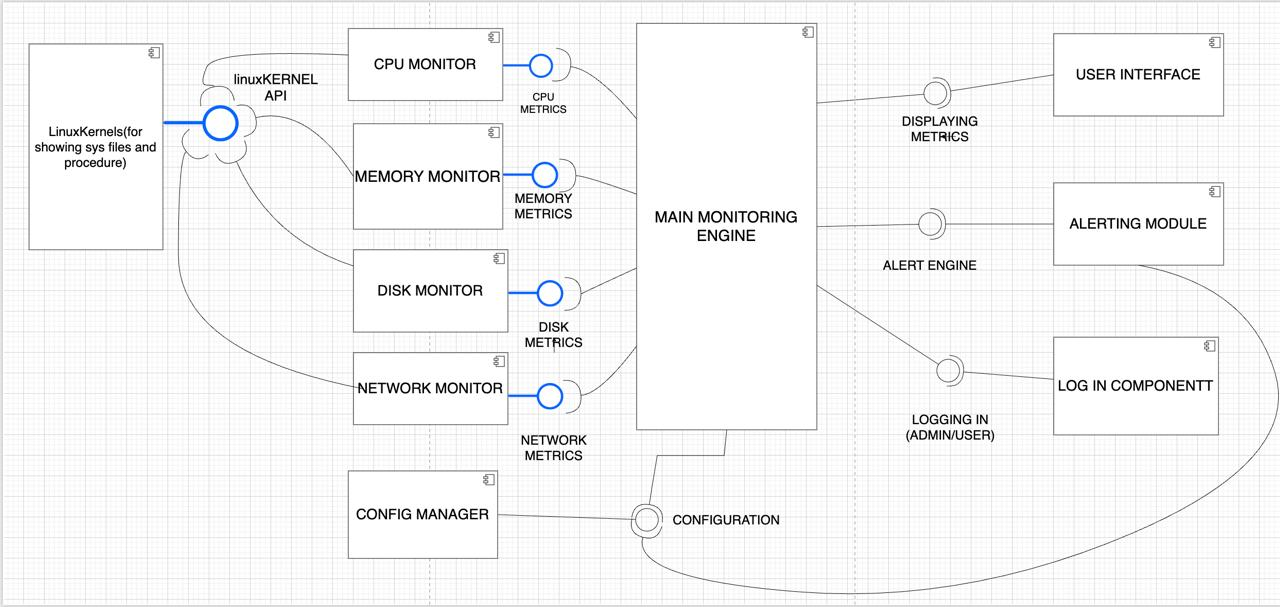
/proc filesystem and OS-specific libraries.

* + Full functionality requires the user to have administrative (root/sudo) privileges.

**3.2 Stakeholders & Concerns**

* **Standard Users:** Usability, clarity of data.
* **Privileged Users (Admins):** Access to granular data, reliability, process management capabilities.
* **Developers:** Modularity, maintainability, clear separation of concerns.
* **QA Engineers:** Testability, traceability to requirements.

**3.3 Component (UML) Diagram**



**3.4 Component Descriptions**

The system is composed of several logical modules identified in the RTM:

* **UI Module:** Renders the Terminal User Interface using the ncurses library and handles user input for sorting and other commands.
* **Auth Module:** Validates user privileges by checking for root or sudo access before allowing privileged operations.
* **Core Monitors (CPU, Memory, Disk, Network):** A set of modules responsible for reading data from specific files in the /proc filesystem (e.g., /proc/stat, /proc/meminfo).
* **Process Manager:** Responsible for listing running processes, gathering their statistics from /proc/[pid], and handling sorting logic.
* **Alerts Module:** Monitors collected metrics and triggers warnings when they cross user-defined thresholds.
* **Config Parser:** Reads and validates the configuration file for custom alert thresholds and other settings.
* **Logging & Export Module:** Manages the creation of structured audit logs and handles the export of monitoring data to CSV/JSON formats.

**3.5 Chosen Architecture Pattern and Rationale**

A **Layered Architecture** has been chosen for this project. This pattern provides a clear separation of concerns, which enhances modularity and maintainability. The layers are:

* **Presentation Layer:** The UI Module, which is responsible for all user interaction.
* **Business Logic Layer:** The Process Manager, Alerts Module, and Config Parser, which implement the core application logic.
* **Data Access Layer:** The Core Monitor modules, which are responsible for interfacing directly with the operating system's /proc filesystem to retrieve raw data.

This layered approach is well-suited for the application's scope and avoids the unnecessary complexity of other patterns like microservices.

**3.6 Technology Stack & Data Stores**

* **Programming Language:** C
* **Compiler:** GCC/minGW or Clang
* **Build System:** Makefile or CMake
* **UI Library:** ncurses for the Terminal User Interface
* **Target OS:** Linux-based distros (e.g., Ubuntu 22.04 LTS) and Unix-based distros
* **Data Stores:** The application is stateless and reads directly from the OS /proc filesystem. It can generate persistent log files (CSV/JSON) on the local disk.

**3.7 Risks & Mitigations**

| Risk | Mitigation | | :--- | :--- | | Inconsistent behavior across different Linux distros / kernel versions | Conduct early and continuous compatibility testing on target distros (Ubuntu, Fedora, Arch). |

| Security vulnerabilities related to privileged access | Implement strict

sudo checks, sanitize all config file inputs, and perform dedicated security testing. |

| Performance degradation on high-load systems | Use dedicated test machines for performance benchmarking and optimize data-reading routines. |

**3.8 Traceability to Requirements**

The architecture directly maps to requirements as detailed in the RTM. For example:

* **SPT-F-001 (Validate admin users):** Mapped to the Auth Module.
* **SPT-F-005 (Display CPU usage):** Mapped to the CPU Monitor module.
* **SPT-F-015 (Export data):** Mapped to the Export/Logging Module.

**3.9 Security Architecture**

The security architecture is designed using the STRIDE threat model:

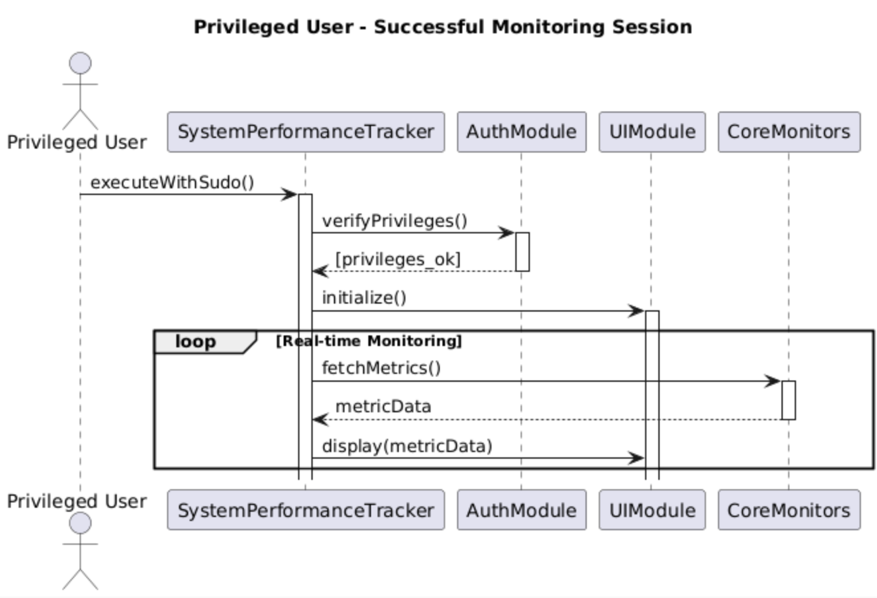
* **Spoofing:** Mitigated by using the operating system's sudo mechanism to validate user identity and privileges. The system itself does not manage user credentials.
* **Tampering:** Mitigated by creating append-only audit logs for privileged actions and setting secure file permissions (e.g., 644) on configuration files to prevent modification by unauthorized users. Input from config files is sanitized.
* **Information Disclosure:** Mitigated through strict Role-Based Access Control (RBAC). Standard users are prevented from viewing privileged system metrics and the environment variables of other processes.
* **Denial of Service (DoS):** Mitigated by designing the tool for low resource overhead (≤ 2% CPU) and ensuring it can recover from transient faults, preventing it from crashing or locking up the system.
* **Elevation of Privilege:** This is the primary threat, mitigated by the Auth Module which explicitly denies access to sensitive functions (like process termination or viewing restricted metrics) unless the application is run with root privileges.

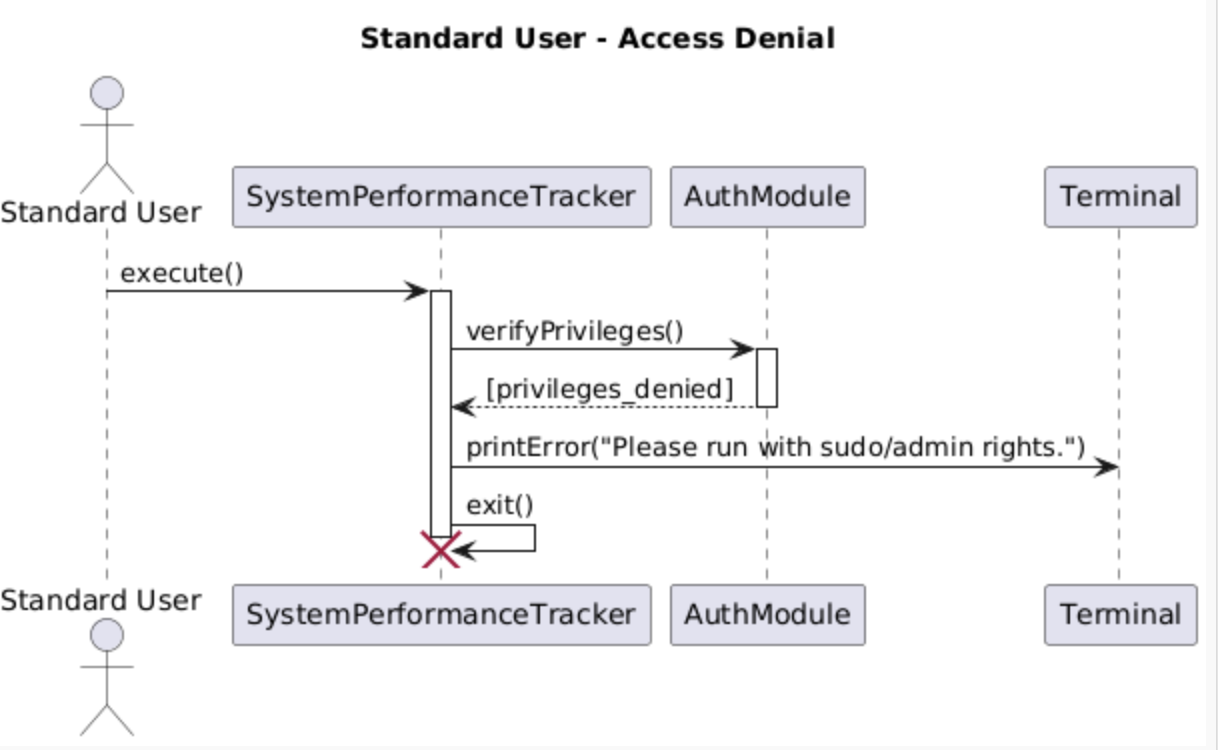
**4. Design**

**4.1 Design Overview**

The system is designed with distinct modules for each core functionality (CPU, memory, etc.). This modular design allows for independent development, testing, and maintenance of each monitoring component.

**4.2 UML Sequence Diagrams**

1. **Sequence 1: Privileged User Monitoring Session.** This diagram would show the sequence: Privileged User executes the program with sudo -> Auth Module verifies root privileges -> UI Module is initialized -> Core Monitor modules are called to fetch data -> Data is displayed in the TUI. 
2. **Sequence 2: Standard User Attempts to Access Privileged Metric.** This diagram would show: Standard User executes the program -> Auth Module detects non-root privileges -> The system prints the "Please run with sudo/admin rights" error message and exits, as required by SPT-F-003.



**4.3 API Design**

As a CLI/TUI application, the "API" is the command-line interface. The system provides two primary modes as required by SPT-F-014:

* **Interactive Mode (TUI):**
  + **Endpoint:** ./spt
  + **Action:** Launches the full-screen terminal user interface for real-time monitoring. Accepts keyboard commands for sorting processes (by CPU, memory, PID).
* **Non-Interactive Report Mode:**
  + **Endpoint:** ./spt --output [csv|json] --file <path>
  + **Action:** Runs the monitoring tool for a single snapshot, exports the data to the specified file in either CSV or JSON format, and then exits.

**4.4 Error Handling, Logging & Monitoring**

* **Error Handling:** The system provides clear error messages for common failures, such as the denial message for insufficient privileges. It is designed to recover from transient metric-read faults within 5 seconds.
* **Logging:** The system produces structured JSON logs with timestamps, log levels, and event IDs. Audit logs for privileged actions are append-only. Logs are automatically rotated based on a configurable policy.

**4.5 UX Design**

The user experience is centered on a clear and readable TUI.

* The UI will support keyboard navigation.
* A high-contrast theme will be available for accessibility.
* The layout will scale correctly for different terminal sizes and system DPI settings (100% and 150%).

**4.6 Open Issues & Next Steps**

* **Next Steps:** Complete implementation of all modules as per the design and proceed with the testing schedule outlined in the STP .
* **Future Enhancements:**
  + Support for historical data analysis.
  + Plugin architecture for custom metric monitoring.

**5. Appendices**

**5.1 Glossary**

* **TTY:** Teletypewriter - A terminal device for user interaction.
* **NFR:** Non-Functional Requirement.
* **RTM:** Requirements Traceability Matrix.