RIHNO – INTRUSION DETECTION SYSTEM USING AI

Software Requirements Specification (SRS)

Field	**Details**
Version	1.0
Prepared By	1. Sakib Dalal 2. Aishwarya Patil
Department	AI & ML
Internship At	intiGrow
Internship Guide	[Guide Name]
Date	February 3, 2026

Contents

1 Introduction

Intrusion Detection System (IDS): An intrusion detection system is a agent or software application that monitors a network or computing systems for malicious activity or policy violations. Any intrusion activity or violation is typically either reported to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system combines outputs from multiple sources and uses alarm-filtering techniques to distinguish malicious activity from false alarms. IDS types range in scope from single computers to large networks. The most common classifications are network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). A system that monitors important operating system files is an example of an HIDS, while a system that analyzes incoming network traffic is an example of an NIDS.

AI-based IDS: An AI-based intrusion detection system leverages artificial intelligence technologies to monitor network traffic and identify unauthorized access or malicious activities. These systems use machine learning to analyze data in real time, allowing them to detect novel threats and adapt to emerging cyber risks more effectively than traditional rule-only methods.

1.1 Purpose

The purpose of this document is to clearly define and detail the software and system requirements for the AI-Based Intrusion Detection System, named **RIHNO**. This system enhances cybersecurity by leveraging artificial intelligence and machine learning techniques to identify, analyze, and respond to both known and unknown security threats in real time.

The proposed IDS is designed to operate effectively across heterogeneous and distributed computing environments, including cloud-based virtual machines (VMs), containerized infrastructures, edge computing agents, and Internet of Things (IoT) nodes. By continuously monitoring network traffic, system logs, and behavioral patterns, the system seeks to detect anomalies, malicious activities, and policy violations with high accuracy and minimal false positives.

This Software Requirements Specification (SRS) serves as a comprehensive reference document throughout the system's lifecycle from design and development to testing, deployment, and maintenance. It establishes a common understanding of system capabilities, constraints, interfaces, and performance expectations among the project team and mentors.

This document is intended for the following audiences:

- **Intern Developers** – to understand functional and non-functional requirements for implementation.
- **Internship Mentors/Supervisors** – to guide and evaluate technical design decisions.
- **Project Evaluators** – to assess compliance with defined objectives and requirements.

1.2 Scope

The AI-Based Intrusion Detection System (**RIHNO**) is designed as a unified security monitoring platform to protect modern hybrid digital infrastructures through real-time visibility and intelligent threat detection. The system architecture, as illustrated in Figure 1, is composed of the following major subsystems:

Agent Layer (Data Collection): Lightweight monitoring agents, built in Go and C++, are deployed across diverse endpoints including cloud virtual machines, containers, local servers, and IoT nodes such as Raspberry Pi and ESP32. A dedicated **Go Producer CLI Tool** (`rihno`) is provided for on-site agent provisioning and lifecycle management. The CLI exposes the following commands: `rihno config` registers a new agent with the backend; `rihno start` begins transmitting host and network logs to the remote server; `rihno start --local` additionally

persists all outgoing logs to a local CSV file; `rihno stop` halts log transmission; `rihno status` prints a runtime summary including the total number of logs transmitted; and `rihno config status` reports whether the agent is currently authenticated. Additional commands will be introduced in future releases. Upon successful registration, the agent name and a unique API key are generated and persisted for future authentication. Registered agents act as **Producers**, continuously collecting host-based metrics (CPU usage, RAM utilization, disk activity, and system temperature) and network-based metrics (traffic IP flow, port activity, and upload/download speeds), then transmitting structured telemetry via TCP to the central ingestion layer.

Central Ingestion Layer (AWS EC2): All telemetry received from the producers is first handled by a **Go Dealer** running on AWS EC2. The Go Dealer performs **Authentication and Verification** of the incoming agent credentials (email, agent name, API key) against records stored in AWS DynamoDB before forwarding the data. Verified telemetry is then dispatched in parallel to two ingestion targets: an **Apache Kafka** cluster for real-time AI-pipeline streaming, and a **PostgreSQL** database for persistent storage and dashboard consumption. This dual-path architecture ensures that real-time threat analysis and persistent data storage operate independently.

AI Detection Pipeline: The Kafka stream is consumed by Python-based consumers (via TCP port 9092) that perform real-time feature engineering, normalization, and scaling using Pandas and Scikit-learn. The processed data is fed into the **AI Decision Engine**, which employs both **Supervised** and **Unsupervised** machine learning models built with Scikit-learn and PyTorch, operating on a Python 3 runtime. A **Rule-Based Decision Engine** complements the ML models to enhance detection accuracy. When the AI Decision Engine determines that an intrusion is present (*"If Intrusion" check*), the **Intrusion Alert Engine** is triggered, generating notifications and persisting alert records.

Web Dashboard and Agent Management: The web-based dashboard is built with React.js, Tailwind CSS, nivo Charts, Axios, and Three.js for interactive visualization. It communicates with the **Node.js + Express.js + Axios** backend via RESTful APIs (HTTP GET) to query PostgreSQL for historical telemetry, alert timelines, agent health, and aggregate security reports. The backend also receives **AI Predictions** and **Intrusion** alerts to relay to the dashboard in real time. User authentication is handled by **AWS Cognito**. The dashboard provides a comprehensive set of features: **agent Management** allows administrators to create, register, edit, and delete monitored agents directly from the UI. A **Network Map** view renders an interactive topology of all registered agents and their interconnections. **Metrics Analysis** pages expose dedicated plots and charts (line, bar, pie, heatmap, and gauge) for CPU, RAM, temperature, network throughput, and other collected telemetry. A **Real-Time Kafka Log Viewer** streams live telemetry messages consumed from Kafka directly into the dashboard so that operators can inspect raw logs as they flow through the system. When an intrusion is detected the **Notification System** can deliver alerts through multiple channels in-app banners, email, and SMS and administrators can manage a list of **Email Recipients** to whom intrusion reports are automatically forwarded. Finally, an **AI Log Analysis** feature sends historical or suspicious log data to an LLM and surfaces a natural-language summary and risk assessment directly inside the dashboard.

1.3 Document Conventions

- **FR:** Functional Requirement.
- **NFR:** Non-Functional Requirement.
- Bold items represent critical modules or shortlisted technologies.

1.4 Intended Audience

The intended audience includes:

- **Internship Mentors and Evaluators** – to review the technical scope, assess requirement completeness, and evaluate the system against project objectives.
- **Intern Developers** – to implement, deploy, and maintain the system, including agent development, backend services, and cloud infrastructure.

2 Overall Description

This section provides a high-level overview of the RIHNO Intrusion Detection System, including its system context, major functions, user categories, operating environment, and underlying assumptions and dependencies.

2.1 Product Perspective

RIHNO is an AI-based Intrusion Detection System designed as a modular, cloud-native security monitoring platform following a distributed architecture. The end-to-end data and control flow as captured in the system architecture diagram (Figure 1) proceeds as follows:

1. **Agent Provisioning:** An administrator uses the **Go Producer CLI Tool** (`rihno`) to register a new monitored endpoint via `rihno config --email <email> --agent_name <n> --api_key <key>`. The CLI submits the credentials to the backend, which verify the registration in **AWS DynamoDB** (via the API Gateway) and returns verification successful message to the operator. The operator can verify authentication status at any time by running `rihno config status`.

2. **Data Collection & Transmission Control:** Once provisioned, the agent is started with `rihno start`, which begins continuous collection of host-level metrics (CPU, RAM, temperature, disk) and network-level metrics (traffic flow, IP addresses, port numbers, upload/download speed) and transmits them via TCP (`POST`) to the **Go Dealer** on AWS EC2. Running `rihno start --local` enables the same remote transmission while also writing every outgoing log entry to a local CSV file for offline backup or analysis. The operator can halt transmission at any time with `rihno stop`, and inspect a runtime summary including the total number of logs sent using `rihno status`. Additional CLI commands will be added in future releases.

3. **Authentication & Verification:** The Go Dealer validates the agent's credentials (email, agent name, API key) against records in AWS DynamoDB. Only verified telemetry is forwarded downstream.

4. **Dual-Path Ingestion:** Verified data is dispatched simultaneously to **Apache Kafka** (for the real-time AI pipeline) and **PostgreSQL** (for persistent storage and dashboard queries). Both services reside within the AWS EC2 environment.

5. **AI Detection Pipeline:** Kafka consumers (TCP port 9092) ingest the stream and perform feature engineering and normalization. The refined data enters the **AI Decision Engine**, which applies both Supervised and Unsupervised models (Scikit-learn + PyTorch on Python 3) and a complementary **Rule-Based Decision Engine**. If the engine classifies an event as an intrusion, the **Intrusion Alert Engine** fires.

6. **Dashboard & Alerting:** The Node.js + Express.js backend queries PostgreSQL and receives AI predictions and intrusion alerts. It serves all data to the React.js dashboard, which exposes the following capabilities: full agent lifecycle management (create, register, edit, delete); an interactive network-map topology rendered; metrics-analysis pages with multiple chart types (line, bar, pie, heatmap, gauge) for CPU, RAM, temperature, and network throughput; a real-time Kafka log viewer that streams live telemetry directly into the UI; a multi-channel notification system (in-app, email, SMS) triggered on intrusion detection; an email-recipient manager for configuring who receives intrusion reports; and an AI Log Analysis panel that forwards log data to an LLM and displays a natural-language risk summary inside the dashboard.

By separating the real-time AI inference path (Kafka) from the persistent data and dashboard path (PostgreSQL), RIHNO ensures that detection latency remains minimal while still providing comprehensive historical visibility.

2.2 Product Functions

The primary functions of the RIHNO system include:

- Agent provisioning via `rihno config`, including agent registration, API-key verification through AWS DynamoDB.
- Authentication-status checking via `rihno config status`, allowing operators to verify agent credentials at any time.
- Agent lifecycle control via `rihno start` and `rihno stop` to begin or halt host and network log transmission to the remote server.
- Local log persistence via `rihno start --local`, which mirrors all outgoing telemetry to a local CSV file for offline backup or analysis.
- Runtime diagnostics via `rihno status`, which prints a summary including the total number of logs transmitted since the agent was started.
- Continuous monitoring of host and network activities across distributed endpoint systems.
- Authentication and verification of agent credentials by the Go Dealer before data ingestion.
- Secure dual-path streaming of verified telemetry to Apache Kafka and PostgreSQL.
- Real-time data preprocessing, feature extraction, and normalization via Kafka consumers (Python, Pandas, Scikit-learn).
- AI-driven detection of known attacks and unknown (zero-day) anomalies using Supervised and Unsupervised ML models (Scikit-learn + PyTorch) and a Rule-Based Decision Engine.
- Intrusion classification and conditional alert activation ("If Intrusion" logic).
- Persistent storage of structured telemetry and security events in PostgreSQL for historical analysis and auditing.
- Alert generation, notification, and persistence for detected intrusions.
- Web-based dashboard agent management: create, register, edit, and delete monitored agents from the UI, with credentials generated via AWS DynamoDB + API Gateway.
- Interactive network-map topology visualization rendered, showing all registered agents and their interconnections.
- Metrics-analysis views with multiple chart types (line, bar, pie, heatmap, gauge) for CPU, RAM, temperature, network throughput, and other collected telemetry.
- Real-time Kafka log viewer that streams live telemetry messages into the dashboard as they are consumed from Kafka.
- Multi-channel intrusion notification system supporting in-app banners, email, and SMS alerts.
- Email-recipient management allowing administrators to configure the list of addresses that receive automated intrusion reports.

- AI Log Analysis feature that sends log data to an LLM and presents a natural-language risk summary and recommendations directly inside the dashboard.
- REST API support for integration with external security tools and services.

2.3 User Classes

The RIHNO system is intended for the following user classes:

- **System Administrators**: Responsible for provisioning agents using the Go Producer CLI Tool, configuring monitoring policies, and managing infrastructure security.
- **Security Analysts**: Monitor alerts, analyze detected threats, and investigate anomalous behavior using dashboards and logs.
- **DevOps Engineers**: Integrate RIHNO into CI/CD pipelines, cloud environments, and container orchestration platforms.

2.4 Operating Environment

RIHNO operates in a hybrid and distributed computing environment designed for high-throughput telemetry processing. The environment includes:

- **Cloud Platform (AWS EC2):** Hosts the Go Dealer, Apache Kafka cluster, and PostgreSQL database. EC2 provides the central compute layer for ingestion, verification, and streaming.
- **Host Operating Systems:** Linux-based distributions for both the lightweight monitoring agents and the backend processing services.
- **Message Broker:** Apache Kafka for real-time message streaming over TCP (Port 9092), serving as the backbone for AI-pipeline data ingestion.
- **Relational Database:** PostgreSQL for persistent storage of structured telemetry, security events, and historical records consumed by the frontend dashboard.
- **Configuration and Metadata Store:** AWS DynamoDB (accessed via API Gateway) for storing agent registrations, agent metadata, API keys, and system configurations.
- **Authentication Service:** AWS Cognito for secure user authentication, authorization, and identity management for the web dashboard.
- **Execution Runtimes:** Python 3 for the AI inference engine and Kafka consumer processing; Node.js for the web backend; Go for the Dealer and CLI Tool.
- **Web Browsers:** Modern browsers (Chrome, Firefox, Safari) with support for JavaScript frameworks and WebSockets for real-time dashboard visualization.
- **Connectivity:** Stable internet and network connectivity for agent–to–dealer communication and remote monitoring.

2.5 Assumptions & Dependencies

The development and operation of RIHNO are based on the following assumptions and dependencies:

- **Agent Permissions:** Target systems and endpoints allow the installation and execution of lightweight monitoring agents with sufficient privileges to access system and network metrics.
- **Network Stability:** Continuous and low-latency network connectivity is available between the distributed agents and the Go Dealer on AWS EC2.
- **AWS Infrastructure:** Core AWS services EC2 for processing, DynamoDB (+ API Gateway) for configuration and credential storage, and Cognito for identity management remain available and stable.
- **Database Availability:** The PostgreSQL database instance remains available and performs well under the expected concurrent read and write load.
- **Model Efficacy:** Machine learning models (Supervised and Unsupervised) are trained on high-quality, representative datasets to ensure detection accuracy and minimize false-positive rates.
- **User Competency:** System administrators and security analysts possess a fundamental understanding of cybersecurity principles, threat landscapes, and system monitoring.
- **Software Ecosystem:** Third-party open-source libraries and frameworks including Apache Kafka, PostgreSQL, Scikit-learn, PyTorch, React.js, and nivo Charts continue to be maintained and receive security updates.

3 System Architecture

Figure 1 presents the complete RIHNO system architecture. The diagram is organized into six logical zones, each described in the subsections that follow.

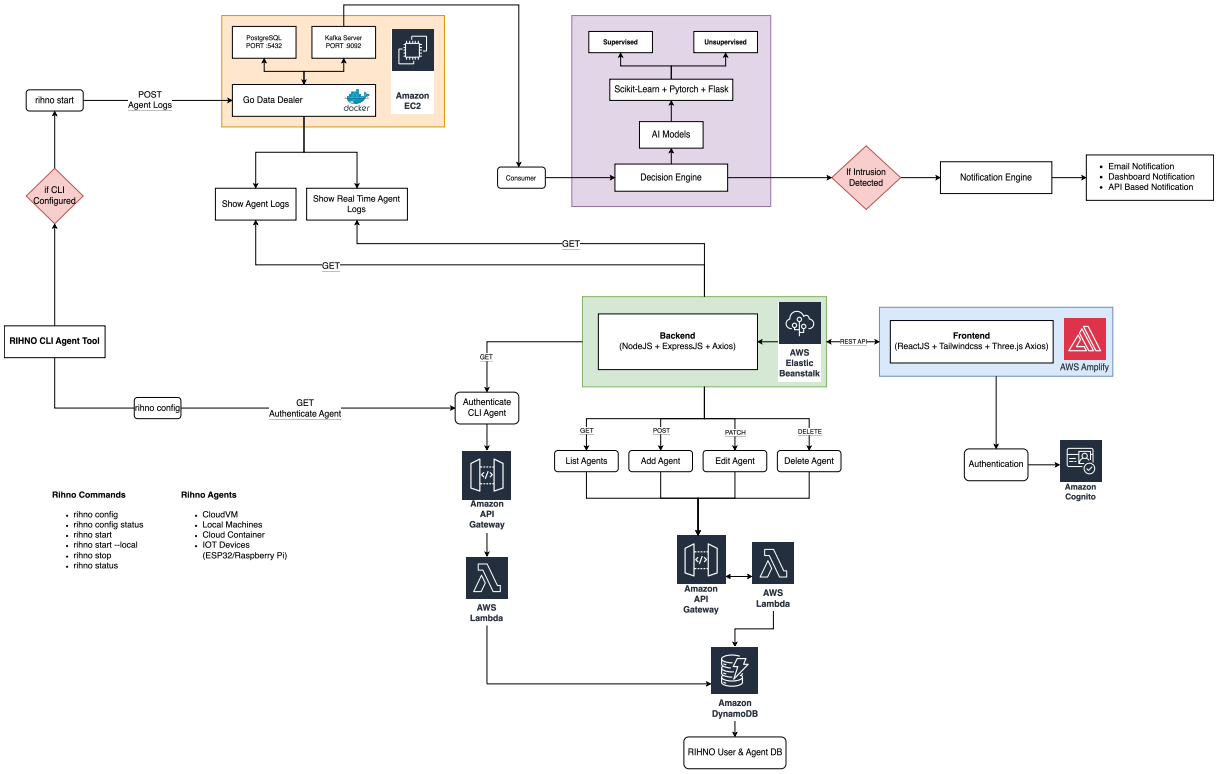

Figure 1: RIHNO System Architecture Diagram

3.1 Agent Layer

The leftmost zone of the architecture comprises the **Go Producer CLI Tool** (`rihno`) and the **Agents/Producers**. The CLI tool is the single entry point for agent registration, transmission control, and runtime diagnostics. Table 1 lists every command currently supported by the tool. *Note: The CLI command set is actively under development. Additional commands will be introduced in future releases to support expanded monitoring, configuration, and debugging capabilities.*

Once registered and started, the agent on that endpoint becomes an active **Producer**, continuously collecting host and network telemetry from one of the supported platform types: Cloud VMs, Containers, Local Servers, or IoT agents (Raspberry Pi / ESP32).

3.2 Central Ingestion Layer (AWS EC2)

All telemetry from the producers arrives at the **Go Dealer**, a lightweight Go service running on AWS EC2. The Go Dealer first performs **Authentication and Verification** by validating the agent's email, agent name, and API key against the records stored in AWS DynamoDB (accessed via the API Gateway). Only after successful verification does the Dealer forward the data in parallel to:

- **Apache Kafka** for real-time streaming to the AI pipeline.
- **PostgreSQL** for persistent storage and dashboard queries.

Table 1: RHINO CLI Command Reference

Command	Category	Description
`rhino config`	Registration	Registers a new agent with the remote backend. Accepts `--email`, `--agent_name`, `--api_key`, and `--agent_type` flags. On success the generated Agent Name and API key are returned to the operator and persisted in AWS DynamoDB.
`rhino config status`	Registration	Prints the current authentication status of the locally registered agent (authenticated or not).
`rhino start`	Transmission	Starts the agent: begins collecting host and network metrics and transmits structured telemetry via TCP to the Go Dealer on AWS EC2.
`rhino start --local`	Transmission	Identical to `rhino start` but additionally writes every outgoing log entry to a local CSV file, enabling offline backup or independent analysis.
`rhino stop`	Transmission	Stops the agent gracefully, halting all metric collection and log transmission to the remote server.
`rhino status`	Diagnostics	Prints a runtime summary of the agent, including the total number of logs transmitted since the last `start`.

3.3 AI Detection Pipeline

Kafka consumers, running on TCP port 9092, ingest the streaming data. The consumed telemetry is preprocessed (feature engineering, normalization, scaling) and then fed into the **AI Decision Engine**. This engine operates within a dashed-border zone that contains:

- A **Rule-Based Decision Engine** that applies static security rules.
- **Supervised** and **Unsupervised** machine learning branches, both implemented via **Scikit-Learn + PyTorch** on a **Python 3** runtime.

When the AI Decision Engine classifies an event as a potential intrusion (*"If Intrusion" check*), the **Intrusion Alert Engine** is triggered, generating and persisting alert records.

3.4 Web Dashboard and Agent Management

The **Web App Backend** (Node.js + Express.js + Axios) serves as the API layer. It pulls historical data from PostgreSQL via GET requests, and also receives *AI Predictions* and *Intrusion* signals to relay to the **Web App Front-end**. The frontend is built with React.js, Tailwind CSS, nivo Charts, Axios, and Three.js, and provides the following features:

agent Management: Administrators can create and register new agents, edit existing agent details (name, configuration), and delete agents that are no longer monitored. Upon creation the system generates a unique agent Name and API key through **AWS DynamoDB + API Gateway** and returns them to the operator.

Network Map: An interactive topology view, displays all registered agents and their network interconnections in a draggable, zoomable layout. This allows security analysts to quickly understand the structure of the monitored infrastructure and trace communication paths between agents.

Metrics Analysis & Visualisation: Dedicated analysis pages present collected telemetry through multiple chart types tailored to different data characteristics: line charts for time-series trends (CPU, RAM, network throughput), bar charts for comparative snapshots, pie charts for

proportion breakdowns (e.g., traffic distribution by agent), heatmaps for spotting anomalies across agents and time ranges, and gauge widgets for at-a-glance status readings (e.g., current CPU or temperature). All plots are powered by nivo Charts and update dynamically as new data arrives from PostgreSQL.

Real-Time Kafka Log Viewer: A dedicated panel in the dashboard subscribes to the live Kafka stream (via WebSockets relayed through the backend) and renders incoming telemetry messages in a scrolling, filterable log table. This gives operators direct visibility into the raw data flowing through the system without needing access to Kafka tooling.

Notification System: When the AI Decision Engine flags an intrusion, the **Intrusion Alert Engine** dispatches notifications through multiple channels simultaneously: an **in-app banner** displayed prominently on the dashboard, an **email alert** sent to all configured recipients, and an **SMS notification** for urgent, eyes-on incidents. Administrators can configure the severity thresholds that trigger each channel independently.

Email Recipient Management: A settings panel allows administrators to maintain the list of email addresses that receive automated intrusion reports. Recipients can be added or removed at any time, and the system validates addresses before persisting them.

AI Log Analysis (LLM-Powered): Operators can select a set of historical or suspicious log entries and submit them to the **AI Log Analysis** feature. The backend forwards the payload to an LLM, which examines the logs for patterns, anomalies, and potential threats. The model's natural-language analysis including a risk assessment and actionable recommendations is rendered directly inside the dashboard, giving analysts an instant expert-level interpretation without manual review.

Authentication across all dashboard operations is handled by **AWS Cognito**.

4 System Features

This section describes the major functional features of the RIHNO AI-Based Intrusion Detection System. Each feature represents a core subsystem responsible for data collection, processing, threat detection, and user interaction.

4.1 Agent Provisioning & Management

RIHNO provides a **Go Producer CLI Tool** (`rihno`) that serves as the single command-line interface for agent registration, transmission control, and runtime diagnostics. The provisioning and operational workflow proceeds as follows.

Registration: An administrator runs `rihno config --email <email> --agent_name <n> --api_key <key>` to verify a new endpoint. The CLI forwards the credentials to the backend, which validates them in **AWS DynamoDB** via the API Gateway. At any later point the operator can run `rihno config status` to check whether the agent is currently authenticated against the remote backend.

Transmission Control: Once registered, the operator starts the agent with `rihno start`. The agent begins collecting host and network metrics and transmits them to the remote server. If local logging is also required, `rihno start --local` performs the same remote transmission while simultaneously writing every log entry to a local CSV file for offline backup or independent analysis. The operator can stop the agent at any time with `rihno stop`.

Diagnostics: Running `rihno status` prints a runtime summary of the agent, including the total number of logs transmitted since the last start. This allows operators to monitor agent health without accessing the web dashboard.

Future Commands: The CLI is designed to be extensible. Additional commands for expanded monitoring, configuration, and debugging capabilities will be introduced in future releases.

From the web dashboard, administrators can **List Agents** to view all currently registered agents and their status, or **Add New Agent/Producer** to register additional endpoints. The dashboard's agent-management interface communicates with the DynamoDB + API Gateway to create and retrieve agent records.

4.2 Agent Deployment & Data Collection

Once provisioned, RIHNO agents are deployed as lightweight processes across heterogeneous environments including cloud virtual machines, containers, local servers, and edge/IoT agents (Raspberry Pi, ESP32). Each agent operates as a **Producer**, continuously collecting:

- **Host-based metrics:** CPU usage, RAM utilization, disk activity, and system temperature.
- **Network-based metrics:** Traffic flow, IP addresses, port numbers, and upload/download speeds.

The collected telemetry is packaged in structured messages and transmitted via TCP (`POST`) to the Go Dealer on AWS EC2.

4.3 Authentication & Verification

Before any telemetry is ingested, the **Go Dealer** performs an **Authentication and Verification** step. It validates the credentials embedded in the agent's transmission (user email, agent name, API key) against the records persisted in **AWS DynamoDB**. Only after successful verification is the data forwarded to Kafka and PostgreSQL. This step prevents unauthorized or spoofed agents from injecting data into the system.

4.4 Data Ingestion & Processing Layer

Verified telemetry is dispatched via two parallel paths:

Kafka Path (AI Pipeline): Apache Kafka serves as the real-time streaming backbone (TCP Port 9092). Python-based consumers subscribe to dedicated topics, performing instantaneous feature engineering, data normalization, and scaling using Pandas and Scikit-learn before passing the refined data to the AI Decision Engine.

PostgreSQL Path (Dashboard & History): Structured telemetry and event records are written to PostgreSQL for long-term retention. The Node.js + Express backend exposes RESTful APIs so the frontend dashboard can query historical trends, alert timelines, agent health snapshots, and aggregate security reports without interfering with the latency-sensitive AI pipeline.

4.5 AI-Powered Threat Detection Engine

The AI-powered threat detection engine is the core intelligence layer of RIHNO. It operates within the AI Detection Pipeline zone and comprises:

- A **Rule-Based Decision Engine** that applies predefined security rules and signatures.
- **Supervised** machine learning models trained on labeled attack datasets.
- **Unsupervised** machine learning models designed to detect novel anomalies without labeled data.

All ML models are built using **Scikit-learn** and **PyTorch**, executing on a **Python 3** runtime. The engine classifies incoming events in real time. If an event is flagged as an intrusion ("If Intrusion" logic), the system triggers the **Intrusion Alert Engine**, which generates notifications and persists alert records to PostgreSQL.

4.6 User Dashboard & APIs

RIHNO provides a feature-rich web-based dashboard that serves as the primary interface for monitoring, analysis, agent management, and incident response. The frontend stack (React.js, Tailwind CSS, nivo Charts, Axios, Three.js) communicates exclusively through the Node.js + Express.js + Axios backend, which abstracts both PostgreSQL queries and external API calls.

agent Management (Create, Register, Edit, Delete): The dashboard exposes a full CRUD interface for monitored agents. Administrators can *create and register* a new agent directly from the UI; the backend calls the AWS DynamoDB + API Gateway to persist the record and generate a unique agent Name and API key. Existing agents can be *edited* updating the agent name, associated email, or configuration or *deleted* when they are decommissioned, with the corresponding DynamoDB record removed.

Network Map: A dedicated view renders an interactive, draggable network-topology graph of all registered agents. Nodes represent individual monitored endpoints; edges represent observed or configured network connections between them. Analysts can zoom, pan, and click individual nodes to drill down into agent-specific metrics and alerts, providing an at-a-glance picture of the monitored infrastructure.

Metrics Analysis & Visualisation Plots: The dashboard includes dedicated analysis pages that present collected telemetry through a variety of chart types, each suited to a different analytical need:

- **Line charts** for time-series trends CPU utilisation, RAM usage, network throughput, and temperature over time.

- **Bar charts** for comparative snapshots side-by-side metric comparisons across agents or across time windows.
- **Pie charts** for proportion breakdowns e.g., share of total network traffic contributed by each agent.
- **Heatmaps** for anomaly spotting a grid of agents versus time slots colour-coded by metric intensity, making outliers immediately visible.
- **Gauge widgets** for at-a-glance status readings current CPU load, temperature, or network latency displayed as a single needle-style indicator.

All plots are rendered by **nivo Charts** and refresh dynamically as new telemetry arrives from PostgreSQL.

Real-Time Kafka Log Viewer: A live-log panel subscribes (via WebSockets relayed through the backend) to the telemetry stream flowing through Apache Kafka. Incoming messages are rendered in a scrolling, searchable, and filterable table in real time. Operators can apply filters by agent, metric type, or timestamp to focus on specific data without needing direct access to Kafka tooling.

Multi-Channel Notification System: When the Intrusion Alert Engine fires an intrusion event, the notification subsystem dispatches alerts through three independent channels:

- **In-app banner** a prominent, colour-coded alert card rendered at the top of the dashboard with severity level and a link to the full incident details.
- **Email notification** an automated report email sent to every address in the configured recipient list, containing the intrusion summary, affected agent, and timestamp.

Email Recipient Management: A settings panel lets administrators maintain the list of email addresses that receive automated intrusion reports. Addresses can be added or removed at any time. The system validates each address format before persisting it, and provides a confirmation toggle so that recipients can be temporarily muted without being deleted.

AI Log Analysis (LLM-Powered): Operators can select one or more historical or suspicious log entries from any dashboard view and submit them to the **AI Log Analysis** feature. The backend packages the selected logs and forwards them to an **LLM**. The model analyses the payload for patterns, correlations, and potential threat indicators, then returns a structured response containing a natural-language risk assessment, an explanation of any anomalies detected, and actionable recommendations. This response is rendered directly inside the dashboard, giving analysts an instant expert-level interpretation and dramatically reducing manual triage time.

REST APIs: The Node.js + Express.js backend exposes RESTful endpoints that underpin every dashboard feature. **AWS Cognito** enforces authentication and role-based access control on all API routes, ensuring that only authorised users and agents can read or modify system resources. These APIs are also available for integration with external security tools and incident-response workflows.

5 External Interface Requirements

This section describes the interfaces between the RIHNO Intrusion Detection System and external hardware, software components, and communication mechanisms.

5.1 Hardware Interfaces

RIHNO interacts with various hardware platforms through lightweight monitoring agents deployed on:

- Cloud-based virtual machines hosted on AWS EC2.
- Physical servers running Linux-based operating systems.
- Containerized environments managed through container runtimes.
- Edge and IoT agents such as Raspberry Pi and ESP32.

The system does not require specialized hardware peripherals. All interactions with hardware components are performed through host operating system interfaces for accessing system metrics, network statistics, and process information.

5.2 Software Interfaces

RIHNO integrates with multiple software components and third-party services:

- **Go Producer CLI Tool (`rihno`):** A command-line application written in Go that provides agent registration (`rihno config`), authentication checks (`rihno config status`), transmission control (`rihno start`, `rihno start --local`, `rihno stop`), and runtime diagnostics (`rihno status`). Registration communicates with AWS DynamoDB via the API Gateway; transmission commands interact with the Go Dealer over TCP. Additional commands will be added in future releases.
- **Go Dealer:** A Go-based service on AWS EC2 responsible for receiving agent telemetry, performing authentication/verification, and forwarding data to Kafka and PostgreSQL.
- **Apache Kafka:** Functions as the primary message broker for the AI detection pipeline, facilitating high-speed transfer of telemetry from the Go Dealer to Python consumers over TCP port 9092.
- **PostgreSQL:** Serves as the relational database for persistent storage of structured telemetry, security events, alerts, and agent metadata. Queried by the Node.js backend to serve the dashboard.
- **AWS DynamoDB + API Gateway:** Stores agent registration records, agent metadata, API keys, and system configurations. Accessed during both CLI registration and Go Dealer verification, as well as agent management from the dashboard.
- **AWS Cognito:** Provides secure user authentication, authorization, and identity management for the web dashboard.
- **Machine Learning Libraries:** Scikit-learn and PyTorch are integrated into the Kafka processing pipeline for feature extraction, model inference (Supervised and Unsupervised), and threat classification on a Python 3 runtime.
- **Web Backend:** RESTful services and WebSocket handlers implemented using Node.js, Express.js, and Axios manage API requests, serve dashboard data from PostgreSQL, and relay AI predictions and intrusion alerts.

- **Frontend Frameworks:** React.js, Tailwind CSS, nivo Charts, Axios, and Three.js for interactive and real-time security dashboard visualization.
- **LLM Integration:** The backend forwards selected log data to an LLM via the Anthropic API for the AI Log Analysis feature. The model returns a natural-language risk assessment and recommendations, which the dashboard renders directly for the operator.

All software interfaces follow standard data formats such as JSON for request and response payloads.

5.3 Communication Interfaces

RIHNO relies on secure and reliable communication interfaces:

- **TCP/IP:** Used for data transmission between agents and the Go Dealer, and between the Go Dealer and both Kafka and PostgreSQL.
- **Kafka Protocol:** Enables publish-subscribe messaging over TCP on Port 9092 for the real-time AI detection pipeline.
- **PostgreSQL Wire Protocol:** Enables structured query and write operations between backend services and the PostgreSQL database.
- **HTTP/HTTPS:** Used for REST API communication between the Node.js backend and the web dashboard, as well as between the CLI/dashboard and the AWS API Gateway (DynamoDB).
- **WebSockets:** Support live data push from the backend to the dashboard frontend, including real-time intrusion alert broadcasts and the continuous stream of Kafka telemetry messages rendered in the live log viewer.
- **Encryption:** Secure communication channels are established using encryption mechanisms to protect data in transit across all interfaces.

6 Non-Functional Requirements

This section defines the non-functional requirements (NFRs) for the RIHNO Intrusion Detection System.

6.1 Performance Requirements

- The system shall process incoming telemetry data in near real-time with minimal latency along the Kafka-to-AI pipeline.
- Kafka consumers shall handle high-throughput data streams without message loss.
- The AI inference engine shall generate intrusion detection results within an acceptable time window suitable for real-time alerting.
- PostgreSQL shall handle concurrent write operations from the Go Dealer and read operations from the dashboard without significant performance degradation.
- Dashboard updates shall reflect new alerts and system metrics with minimal delay.
- The Go Dealer shall authenticate and verify agent credentials with negligible latency to avoid blocking data ingestion.

6.2 Scalability Requirements

- The system shall support horizontal scaling to accommodate an increasing number of monitored agents and agents.
- Kafka topics and partitions shall be scalable to handle growing data volumes within the AI detection pipeline.
- The PostgreSQL database shall be scalable to accommodate growing volumes of historical telemetry and event data.
- AWS DynamoDB shall scale automatically to handle growing numbers of registered agents and configuration records.
- The backend services shall support scaling in cloud environments without service interruption.

6.3 Reliability and Availability

- The system shall ensure high availability of data ingestion and processing components across both the Kafka and PostgreSQL paths.
- Temporary network failures shall not result in permanent data loss on either ingestion path.
- The system shall recover gracefully from component failures, including failures in the Go Dealer, Kafka, PostgreSQL, or backend services.

6.4 Security Requirements

- The Go Dealer shall authenticate every incoming agent connection using credentials stored in DynamoDB before forwarding telemetry.
- Authentication and authorization mechanisms (AWS Cognito) shall prevent unauthorized access to the dashboard and API resources.
- Sensitive security data stored in PostgreSQL and DynamoDB shall be protected against unauthorized disclosure and tampering.
- API keys generated during agent registration shall be treated as secrets and handled accordingly.

6.5 Usability Requirements

- The web dashboard shall provide an intuitive and user-friendly interface for viewing alerts, agent status, and historical trends.
- The agent-management interface shall allow administrators to create, register, edit, and delete agents through a consistent and clearly labelled CRUD workflow that requires minimal steps at each stage.
- The interactive network-map view shall be navigable by zooming, panning, and clicking individual agent nodes, with drill-down details loading without a full page reload.
- Metrics-analysis pages shall allow users to switch between chart types (line, bar, pie, heatmap, gauge) and apply time-range or per-agent filters without requiring technical knowledge.
- The real-time Kafka log viewer shall render incoming messages with minimal latency and provide search and filter controls so that operators can locate specific entries quickly.
- Intrusion notifications shall be clearly colour-coded by severity and displayed prominently.
- The email-recipient management panel shall validate addresses on entry and provide an obvious way to add, mute, or remove recipients.
- The AI Log Analysis panel shall clearly indicate when an LLM request is in progress, present the returned risk assessment in plain language, and allow the operator to copy or export the analysis.
- The Go Producer CLI Tool shall provide clear usage instructions and error messages for every command.
- The system shall require minimal training for security analysts and administrators.

6.6 Maintainability Requirements

- The system shall follow a modular design to facilitate maintenance and upgrades.
- Machine learning models (Supervised and Unsupervised) shall be updatable without requiring system downtime.
- PostgreSQL schema changes shall be managed through versioned migration scripts.
- DynamoDB table structures shall be documented and versioned to support safe updates.

6.7 Portability Requirements

- Monitoring agents shall be portable across different Linux-based environments.
- The Go Dealer and backend services shall be deployable on both cloud and on-premise infrastructures.

7 Other Requirements

7.1 Ethical & Legal Concerns

The RIHNO system is designed with careful consideration of ethical responsibilities and legal obligations related to cybersecurity monitoring and data handling.

- The system shall collect only security-relevant telemetry required for intrusion detection and system monitoring.
- User privacy shall be respected by avoiding unnecessary collection of personally identifiable information (PII). Agent registration collects only the email, agent name, and API key necessary for authentication.
- All collected data, including data persisted in PostgreSQL and DynamoDB, shall be handled in accordance with applicable data protection and privacy regulations.
- Access to monitoring data and alerts shall be restricted to authorized users only, enforced via AWS Cognito.
- The system shall ensure transparency in alert generation to avoid unfair or biased decision-making by AI models.
- Logs and audit trails shall be maintained to support accountability and forensic analysis.

7.2 Limitations

Despite its capabilities, the RIHNO Intrusion Detection System has certain limitations:

- The accuracy of threat detection depends on the quality and representativeness of training data used for Supervised and Unsupervised ML models.
- False positives or false negatives may occur, particularly for novel or highly sophisticated attacks.
- Real-time detection performance along the Kafka pipeline may be affected under extremely high data ingestion loads.
- PostgreSQL query performance for dashboard views may degrade if historical data volumes grow significantly without appropriate indexing or archival strategies.
- The system requires stable network connectivity between agents and the Go Dealer on AWS EC2.
- If DynamoDB is temporarily unavailable, agent authentication and new agent registration may be delayed.
- RIHNO provides detection and alerting capabilities but does not automatically remediate or block attacks.
- The system's effectiveness may vary across different deployment environments and configurations.

8 Appendix

8.1 Acronyms

Acronym	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Deployment
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
HIDS	Host-based Intrusion Detection System
IDS	Intrusion Detection System
IoT	Internet of Things
LLM	Large Language Model
ML	Machine Learning
NFR	Non-Functional Requirement
NIDS	Network Intrusion Detection System
PII	Personally Identifiable Information
REST	Representational State Transfer
SRS	Software Requirements Specification
SMS	Short Message Service
TCP/IP	Transmission Control Protocol / Internet Protocol
VM	Virtual Machine
WebSocket	A full-duplex communication protocol over a single TCP connection, used for real-time push from server to client

8.2 Tools & Technologies Used

Category	Details
Programming Languages	Go (Dealer, CLI Tool, Agents); C++ (Agents); Python 3 (AI Engine, Kafka Consumers); JavaScript/Node.js (Backend)
CLI Tool	Go Producer CLI Tool (`rihno`) `config` (registration), `config status` (auth check), `start` / `start --local` (transmission with optional local CSV), `stop` (halt), `status` (diagnostics). Extensible; more commands planned.
Ingestion Service	Go Dealer authenticates agents and routes telemetry to Kafka and PostgreSQL
Streaming Platform	Apache Kafka real-time data bus for AI pipeline (TCP Port 9092)
Relational Database	PostgreSQL persistent storage for telemetry, events, alerts, agent metadata
Configuration Store	AWS DynamoDB + API Gateway agent registrations, API keys, system configs
ML Libraries	Scikit-learn (feature engineering, Supervised/Unsupervised models); PyTorch (deep learning models)
Backend Frameworks	Flask (Python stream processing); Node.js + Express.js + Axios (web backend, API layer)
Frontend Frameworks	React.js (UI); Tailwind CSS (styling); nivo Charts (line, bar, pie, heatmap, gauge plots); Axios (HTTP);
LLM Integration	AWS Bedrock powers the AI Log Analysis feature; receives log payloads from the backend and returns natural-language risk assessments and recommendations
Authentication	AWS Cognito identity management and role-based access control
Cloud Platform	Amazon Web Services (AWS) EC2 for compute, managed services for scaling
Version Control	Git and GitHub for collaborative development, source code management, and CI/CD

9 References

References

[1] Wikipedia contributors, "Intrusion Detection System," [Online]. Available: https://en.wikipedia.org/wiki/Intrusion_detection_system (Accessed: Jan. 19, 2026).

[2] Faddom, "AI Intrusion Detection: Components, Challenges and Best Practices," [Online]. Available: https://faddom.com/ai-intrusion-detection-components-challenges-and-best-practices/ (Accessed: Jan. 19, 2026).

[3] The Go Authors, "Documentation – The Go Programming Language," [Online]. Available: https://go.dev/doc/ (Accessed: Jan. 19, 2026).

[4] Microsoft, "C++ Documentation – Microsoft Learn," [Online]. Available: https://learn.microsoft.com/en-us/cpp/cpp/?view=msvc-170 (Accessed: Jan. 19, 2026).

[5] Wikipedia contributors, "Raspberry Pi Foundation," [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi_Foundation (Accessed: Jan. 19, 2026).

[6] Apache Software Foundation, "Kafka – Getting Started: Introduction," [Online]. Available: https://kafka.apache.org/41/getting-started/introduction/ (Accessed: Jan. 19, 2026).

[7] Scikit-learn Developers, "Scikit-learn: Machine Learning in Python," [Online]. Available: https://scikit-learn.org/stable/index.html (Accessed: Jan. 19, 2026).

[8] PyTorch Contributors, "PyTorch Documentation," [Online]. Available: https://docs.pytorch.org/docs/stable/index.html (Accessed: Jan. 19, 2026).

[9] Amazon Web Services, "AWS DynamoDB Developer Guide," [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ (Accessed: Jan. 19, 2026).

[10] Amazon Web Services, "Amazon Cognito Developer Guide," [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/ (Accessed: Jan. 19, 2026).