Enhancing Resilience of ROS-Based Husky UGV to Sensor Data Poisoning

Joseph Tshisau
Dept. of Computer Science
Texas Tech University
Lubbock, TX
tmusangu@ttu.edu

Noah Kueng
Dept. of Computer Science
Texas Tech University
Lubbock, TX
nkueng@ttu.edu

Delphin Iradukunda
Dept. of Computer Science
Texas Tech University
Lubbock, TX
diraduku@ttu.edu

Omotoyosi Adams
Dept. of Computer Science
Texas Tech University
Lubbock, TX
omoadams@ttu.edu

John Heitzman
Dept. of Computer Science
Texas Tech University
Lubbock, TX
john.m.heitzman@ttu.edu

***Abstract*—Autonomous vehicles depend on accurate sensor data to navigate safely. When inputs are corrupted or falsified (*data poisoning*), vehicles may make unsafe decisions. This capstone project analyzes logged datasets from a ClearPath Husky UGV running ROS, reconstructs nominal vehicle behavior, and explores defensive techniques that detect, resist, and recover from falsified sensor inputs. Using ROS bag files, CAN bus logs, and system outputs, we (i) baseline normal operation, (ii) identify anomalies and correlations, and (iii) evaluate mitigation strategies (e.g., cross-sensor validation, temporal consistency checks, trust scoring). While the testbed is a Husky UGV, methods target generalizability to other ROS-based platforms.**

***Index Terms*—autonomous systems, ROS, Husky UGV, data poisoning, anomaly detection, sensor fusion, resilience**

I. Introduction

Autonomy pipelines are vulnerable when adversaries or faults corrupt perception inputs. Our objective is to improve the robustness of a ROS-based Husky UGV to falsified sensor data by learning normal patterns, detecting deviations, and mitigating the effect of suspect inputs while maintaining safe behavior. We analyze provided datasets (ROS bag files, CAN bus logs, system-level outputs) to reconstruct behavior, quantify normal vs. abnormal patterns, and prototype defenses that preserve mission safety under perturbed inputs.

II. Background and Problem Statement

In data poisoning, falsified inputs (e.g., camera range suddenly shrinking from 20 ft to 3 ft without corresponding motion) can trigger evasive maneuvers or unsafe paths. Rather than constructing exploits, we assume a successful injection and study the vehicle's response and defenses. Because many UGV/UAV systems run ROS, improving resilience on Husky has broader applicability.

III. System Overview

A. Product Overview

The deliverable is a set of analysis tools, models, and recommendations to increase resilience of ROS-based vehicles against sensor data poisoning. Outcomes include: (1) parsers and replay utilities for ROS/CAN logs; (2) empirical characterization of normal behavior and correlations; (3) prototype anomaly detectors and mitigation strategies; (4) evaluation on recorded datasets.

B. Product Functionality

The system will:

- Reconstruct vehicle behavior from logs to establish a baseline.
- Identify anomalies, outliers, and cross-sensor correlations.
- Simulate poisoning via labeled perturbations in replayed data.
- Evaluate mitigations (cross-validation among sensors, temporal consistency, trust scores) and provide recommendations.

C. Operating Environment

Software: ROS (bag replay), Python (NumPy, pandas, scikit-learn, PyTorch/TensorFlow), visualization (RViz, Matplotlib).
Hardware: Lab workstations with adequate storage/compute; no physical Husky required.
Targets: ROS-based UGV/UAV platforms.

D. Design and Implementation Constraints

- **Large datasets:** up to $\sim$250 GB uncompressed; storage and preprocessing cost.
- **Scope limits:** no attack construction; evaluation via assumed injections.
- **Timeline:** capstone schedule requires prioritizing tractable techniques.
- **Bootstrapping:** leverage prior parsing utilities to avoid reimplementing ingestion.

E. Assumptions and Dependencies

Assumptions: Datasets reflect nominal Husky operation; simulated perturbations approximate impactful falsifications; methods generalize across ROS systems.
Dependencies: Full datasets and prior intern code; ROS toolchain; faculty/mentor guidance.

IV. External Interface Requirements

A. User Interfaces

Web Dashboard: Reactive UI (e.g., React/Vue) for live and historical views, role-based access, filtering, anomaly dashboards.
CLI: Python/Go utilities for ingestion, preprocessing, batch analysis.
APIs: REST/GraphQL (OpenAPI 3.0); export to CSV/JSON/XML/Parquet with secure links.

B. Hardware Interfaces

Robot comms over Ethernet/Wi-Fi; ROS 2 DDS transport; storage to POSIX filesystem and cloud (S3/GCS/Azure); sensor integration (LiDAR, GPS, camera, IMU) via ROS 2 topics; PTP time sync; CPU/GPU acceleration.

C. Software Interfaces

Linux tracing (strace/eBPF/auditd); databases (PostgreSQL/MySQL/MongoDB); search (Elasticsearch/OpenSearch); ROS 2 compatibility and gRPC microservices; batch/stream processing (Spark/Flink); Python data/ML stack.

D. Communication Interfaces

TCP/UDP/QUIC; TLS 1.3; JSON/Protobuf; Avro/Parquet; WebSocket/gRPC streaming; Kafka/MQTT telemetry; SFTP/SCP/HTTPS transfers; OCI registries.

V. Functional Requirements

A. Data Collection

- FR-001: Collect system call data in real-time (e.g., strace) with mode awareness (autonomous/manual/test).
- FR-003: Capture timestamp, PID, duration, return code, arguments; support concurrent streams (planner, LiDAR, cameras, GPS).

B. Data Processing and Analysis

- FR-005: Identify patterns/anomalies; categorize calls (e.g., `clock_gettime`, `futex`, `read`/`write`, `select`, `recvfrom`, `brk`, `nanosleep`).
- FR-009: Compare across modes; quantify distributions (e.g., timing/synchronization/I/O shares).
- FR-012: Correlate system calls with behaviors and sensor topics (LiDAR, navigation).

C. Visualization and Reporting

- FR-013–FR-016: Real-time and historical dashboards for frequency, timing, component activity and trends.
- FR-017–FR-020: Generate reports with findings, recommendations; export data; summarize key statistics.

VI. Use Case Model

A. Primary Use Cases

UC-001: Real-Time Monitoring (Administrator views live syscall streams; alerts on critical issues).
UC-002: Historical Analysis (Analyst filters time ranges; system produces stats/visuals).
UC-003: Performance Optimization (Engineer finds bottlenecks, validates improvements).
UC-004: Cross-Mode Comparison (Scientist compares autonomous/manual/test behaviors).

B. Secondary Use Cases

UC-005: Data Export and Sharing (CSV/JSON/PDF).
UC-006: System Configuration (RBAC-protected parameter editing).

VII. Non-Functional Requirements

A. Performance

Process $\geq$1.2M calls in $\leq$5 minutes; stream $\geq$11,200 calls/s; sub-second dashboard queries; concurrent session analysis; efficient storage, indexing, archival; robust high-bandwidth, low-latency networking; parallel and incremental analytics with caching.

B. Safety and Security

Encrypt data in transit/at rest; RBAC and MFA; audit logs; defend against common vulns; TLS; input validation; regular patches; do not interfere with robot safety systems; e-stop; health monitoring; safe-mode testing; privacy compliance (GDPR/CCPA), anonymization, retention, audit support.

C. Other Qualities

Scalability (horizontal scale, load balancing, failover, autoscaling); backup/recovery (daily backups, PITR, off-site copies, RTO $\leq$4h); logging/monitoring (aggregation, metrics, alerts, dashboards); API/third-party integrations (SDKs, webhooks, rate limiting/versioning, cloud platforms, Jupyter/R).

VIII. Evaluation Plan (Overview)

We will (1) baseline nominal behavior; (2) define perturbations representative of falsification; (3) measure detection (precision/recall, false alarms), latency, and safety impact; and (4) assess graceful degradation (slow/stop policies) and explainability of flags.

IX. Conclusion

This work packages practical tools and empirical evidence for defending ROS-based autonomy against falsified inputs. By combining behavior reconstruction, anomaly detection, and mitigation policies, we aim to preserve safe operation despite corrupted sensor streams and to generalize lessons to similar platforms.

Acknowledgment

We thank our advisors and collaborators for access to datasets and prior parsing utilities that accelerated our analysis.