

SecureGPS

GPS Spoofing Detection System

Team Leader: Daniel Bikowicz

Product Owner: Helen Gomez

Nahahme Bar'Ysrael - Omar Adel Alali - Matthew Clayton

Faculty Advisor: Prof. Masoud Sadjadi

Spring 2026 - Senior Cybersecurity Capstone - CIS 4951

College of Engineering & Computing

Florida International University

1. Executive Summary

SecureGPS is a real-time GPS spoofing detection system designed to identify and analyze false location data without requiring any hardware modifications. GPS spoofing is a growing cybersecurity threat where attackers broadcast fake signals to manipulate a device's perceived location, and because civilian GPS lacks authentication, most systems blindly trust this data.

This project focuses on solving that gap by building a server-side detection platform that evaluates incoming GPS telemetry in real time. Devices send data over HTTP, and the backend applies multiple detection methods including speed validation, geofence checks, signal quality analysis, and directional consistency. Each event is then assigned a severity level and cryptographically signed using HMAC-SHA256 to ensure integrity.

The system also includes a SOC-style dashboard that visualizes device activity, alerts, and forensic data, allowing users to monitor and investigate suspicious behavior. Based on testing with simulated attack scenarios, the system achieved 100% detection accuracy with 0% false positives and under 50 ms latency per event .

Overall, SecureGPS provides a scalable, software-only solution that improves trust in GPS-based systems across industries like logistics, transportation, and critical infrastructure.

2. Problem & Requirements

GPS spoofing allows attackers to manipulate a device's location, speed, and time by broadcasting counterfeit signals. This is especially dangerous because civilian GPS systems do not include any built-in authentication, meaning receivers cannot verify whether a signal is legitimate. On top of that, these attacks are highly accessible since they can be carried out using low-cost software-defined radio (SDR) hardware, often around \$300. Many real-world systems, such as fleet tracking platforms and ride-share applications, operate under the assumption that GPS data is always valid. Because of this, spoofed data is often accepted as legitimate, creating a major security vulnerability.

The main stakeholders for this system include fleet management companies, ride-share platforms, logistics and delivery services, law enforcement or monitoring agencies, and security analysts working in SOC environments. These groups rely heavily on accurate GPS data for operations, safety, and decision-making, making them particularly vulnerable to spoofing attacks.

From a functional standpoint, the system is designed to ingest GPS telemetry through HTTP POST requests, detect spoofing behavior using multiple independent detection methods, and assign severity levels such as Allow, Low, Medium, High, or Critical. It also stores all processed events with cryptographic integrity verification and presents the results through a real-time dashboard that includes alerts and device tracking. Additionally, the system supports forensic verification of stored data to ensure that events have not been tampered with after ingestion.

In terms of non-functional requirements, the system must maintain a detection latency of under 50 milliseconds per event while achieving high accuracy and minimizing false positives. It should be capable of scaling

to handle multiple devices simultaneously, operate without requiring any hardware modifications, and provide real-time visualization of events and alerts.

There are several constraints associated with the current implementation. As an MVP, the system uses SQLite, which introduces limitations in terms of concurrency and scalability. It also does not include authentication, as the focus is on demonstrating detection capabilities rather than full system security. Additionally, the system is currently designed to run locally and is not deployed in a cloud environment.

Success for this project is defined by the system's ability to detect all simulated spoofing scenarios accurately, avoid false positives during normal operation, provide real-time visibility into alerts, and maintain data integrity through cryptographic signing. However, there are still risks to consider, including the possibility of false positives in real-world environments, variability in GPS signal quality, scalability limitations due to SQLite, and the lack of authentication and security hardening in this prototype stage.

The prioritized backlog for this project focused on building the core detection engine first, followed by data integrity, storage, and then the user-facing dashboard and monitoring features.

- GPS Telemetry Ingestion Endpoint
 - Developed the /ingest API endpoint to receive GPS telemetry via HTTP POST requests. This included defining the input schema (device ID, timestamp, latitude, longitude, HDOP) and validating incoming data using Pydantic to ensure consistent and reliable ingestion.
- Speed-Based Spoofing Detection (Teleportation Detection)
 - Implemented logic using the Haversine formula to calculate distance between consecutive GPS points and derive speed. Any speed exceeding realistic thresholds (> 60 m/s) is flagged as a potential spoofing attack.
- Geofence Validation
 - Defined authorized geographic boundaries (South Florida operating zone) and implemented checks to flag any GPS readings that fall outside these bounds. This helps detect drift or relocation-based spoofing attacks.
- HDOP Anomaly Detection
 - Integrated detection based on GPS signal quality by analyzing HDOP values. Readings with unnaturally low HDOP (< 0.5), which indicate artificially “perfect” signals, are flagged as suspicious.
- Bearing Reversal Detection
 - Calculated directional changes between consecutive GPS points and flagged cases where a device appears to reverse direction ($> 160^\circ$ change) at high speeds, which is physically unrealistic for normal vehicle movement.
- Severity Scoring System
 - Designed a rule-based severity classification system that aggregates detection results and assigns levels (Allow, Low, Medium, High, Critical) based on the number and type of anomalies detected.
- HMAC-Based Event Signing (Data Integrity)
 - Implemented cryptographic signing of each event using HMAC-SHA256 to ensure that stored telemetry data cannot be altered without detection, supporting forensic validation.
- SQLite Event Storage

- Designed and implemented a lightweight database schema to store all events, including device data, detection results, severity levels, and signatures. Chosen for simplicity and ease of deployment in an MVP environment.
- Real-Time Dashboard Map (Leaflet.js)
 - Built a live monitoring interface using Leaflet.js to visualize device locations on a map with color-coded markers (green for normal, red for suspicious), including clustering for scalability.
- Device Tracking and History
 - Implemented per-device state tracking to maintain previous GPS points, enabling calculations for speed and bearing, as well as displaying historical movement paths on the dashboard.
- Demo Simulation Script
 - Built a Python-based simulation script (send_scenarios.py) to generate realistic GPS data and spoofing attack scenarios, including teleportation, geofence drift, HDOP spoofing, and bearing reversal.

The project backlog, user stories, and sprint grooming artifacts were maintained throughout development and are available here: [Backlog Documentation](#).

3. System Overview & Architecture

SecureGPS is designed as a client-server system that processes GPS telemetry in real time and detects spoofing behavior through a centralized backend. GPS-enabled devices send structured telemetry data over HTTP to the backend, where each incoming request is immediately evaluated by the detection engine. The system does not rely on any modifications to the GPS hardware itself, which allows it to work with any device capable of transmitting location data. As shown in Figure 1, the system follows a centralized architecture where telemetry is processed, stored, and visualized in real time.



Figure 1. SecureGPS System Architecture Diagram

At the core of the system is a FastAPI-based backend that handles data ingestion, validation, and processing. When telemetry is received, the system compares the new data against the previous state of that device, which is stored in memory. This enables the detection engine to calculate speed, direction changes, and other behavioral patterns across consecutive GPS points. Each event is then evaluated using multiple independent detection methods, and the results are aggregated into a severity classification. After processing, the event is cryptographically signed using HMAC-SHA256 and stored in a SQLite database to ensure data integrity and support forensic analysis.

The stored data is exposed through a set of REST API endpoints, which are continuously polled by a frontend dashboard. The dashboard provides a SOC-style interface that visualizes device activity on a real-time map and displays alerts based on detected anomalies. This allows users to monitor multiple devices simultaneously, investigate suspicious behavior, and interact with alerts through acknowledgment workflows. The system is designed to operate with low latency by performing all detection logic in memory and minimizing database operations in the critical processing path.

Key Technologies and Components

- Backend Framework: FastAPI (Python) for asynchronous API handling
- Server: Uvicorn ASGI server for running the application
- Data Validation: Pydantic for enforcing structured input schemas
- Database: SQLite for lightweight event storage
- Frontend: HTML, CSS, and JavaScript for the dashboard interface
- Mapping Library: Leaflet.js for real-time geospatial visualization
- Templating Engine: Jinja2 for server-side rendering
- Cryptography: HMAC-SHA256 for event integrity verification

API Surface (Core Endpoints)

- /ingest (POST) – Receives GPS telemetry and runs detection logic
- /events (GET) – Returns recent events
- /alerts (GET) – Returns only flagged/suspicious events
- /stats (GET) – Provides system metrics (events, alerts, devices)
- /verify/{id} (GET) – Verifies event integrity using HMAC signature
- /health (GET) – Checks system status

The system uses a SQLite database to store processed GPS events. Each event record includes the device ID, timestamp, latitude, longitude, detection decision, severity level, and a cryptographic signature. Additional fields track whether an alert has been acknowledged. This structure supports both real-time monitoring and post-event forensic analysis, allowing the system to verify whether any stored data has been modified after ingestion.

5. Implementation Notes (O2)

The SecureGPS system was implemented as a modular Python-based application using FastAPI as the backend framework. The overall design focused on keeping the system lightweight, easy to run, and clearly structured, while still supporting real-time processing and detection of GPS spoofing attacks. The backend handles all core functionality, including data ingestion, detection logic, event storage, and API responses, while the frontend dashboard is responsible for visualization and user interaction.

The repository was organized to separate core components of the system, including the API, detection logic, templates, and simulation tests. This allows for easier debugging, scalability, and future enhancements. The main application logic resides in the FastAPI application file, where endpoints are defined and detection methods are executed. Supporting files handle database interactions and data modeling, while the frontend templates render the dashboard interface. Additional details on project setup, installation, and usage are documented in the README.md file, which provides step-by-step instructions for running the application, accessing the dashboard, and interacting with the system's API endpoints. Figure 2 further illustrates the internal repository structure, showing how the system is organized and separated across the project.

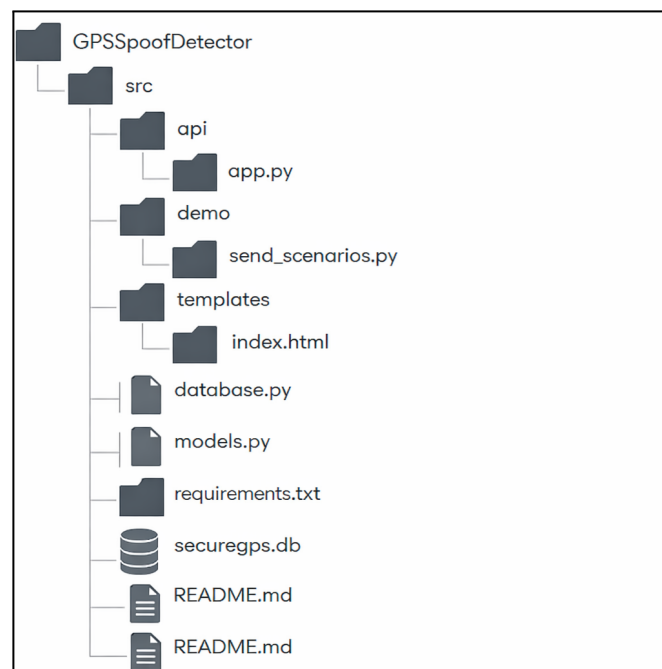


Figure 2. SecureGPS Repository Structure Graphic

The system follows a stateful detection approach, where each device's previous GPS reading is stored in memory and used to calculate speed and bearing changes for new incoming data. This design enables real-time analysis without requiring expensive database queries during processing. Detection methods were implemented as independent checks within the ingestion pipeline, allowing multiple anomalies to be evaluated simultaneously and combined into a final severity decision.

Several design decisions were made to balance simplicity and functionality. SQLite was chosen as the database because it requires no setup and supports quick development, even though it introduces limitations for concurrency and scalability. FastAPI was selected for its asynchronous capabilities, built-in validation through Pydantic, and automatic API documentation. The frontend was intentionally kept simple using HTML, CSS, and JavaScript without a framework, which reduced complexity and allowed the focus to remain on backend detection logic.

The system also incorporates cryptographic integrity through HMAC-SHA256 signing of each event at the time of ingestion. This ensures that any modifications to stored data can be detected during verification. Additionally, a polling mechanism was used for real-time updates on the dashboard, where the frontend requests new data from the backend every few seconds. While this approach is less efficient than WebSockets, it simplifies

implementation and is sufficient for an MVP. The following coding conventions and design tradeoffs were applied throughout the implementation.

Coding Conventions and Patterns

- Modular structure separating API, data handling, and frontend
- Clear endpoint naming following REST conventions
- Consistent data validation using Pydantic models
- Reusable utility functions for calculations (Haversine, bearing)
- In-memory state tracking for per-device computations

Key Tradeoffs

- SQLite chosen over PostgreSQL for simplicity and ease of setup
- Polling used instead of WebSockets for real-time updates
- No authentication implemented to prioritize core detection functionality
- Lightweight frontend used instead of a full framework

6. Testing & Evaluation (O3)

The SecureGPS system was evaluated using a structured, scenario-based testing approach designed to simulate both normal GPS behavior and multiple types of spoofing attacks. Testing focused on validating the accuracy of the detection engine, measuring system performance, and ensuring reliability under continuous data ingestion. Rather than relying solely on unit testing, the evaluation emphasized realistic end-to-end scenarios that reflect how the system would behave in a real-world environment.

A simulation script was developed, instead, to generate GPS telemetry across multiple devices, allowing controlled testing of different attack types. These scenarios included normal driving behavior as a baseline, as well as spoofing cases such as teleportation attacks, geofence violations, HDOP signal anomalies, and rapid bearing reversals. Each scenario had a known expected outcome, which allowed for direct comparison between actual system behavior and expected detection results.

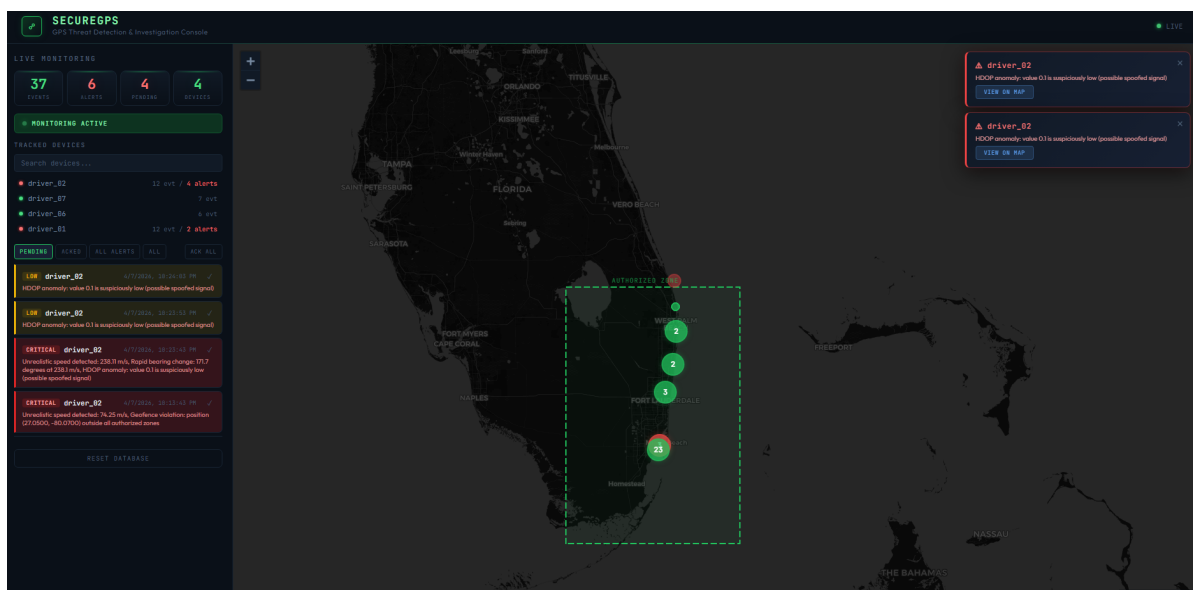


Figure 3. Example of spoofing detection during simulation, with real-time alerts and flagged events

Testing Strategy

- End-to-End Scenario Testing: Simulated real GPS data flows from multiple devices to validate full system behavior, from ingestion to dashboard visualization.
- Detection Validation: Verified that each detection method (speed, geofence, HDOP, bearing) correctly identified its corresponding spoofing scenario.
- Integration Testing: Tested interaction between components, including API endpoints, detection logic, database storage, and frontend updates.
- Manual Validation and Observation: Observed dashboard outputs and alert behavior in real time to confirm correct severity classification and system responses.

The system was tested using approximately 85 GPS readings across 8 simulated devices. These tests included both normal and malicious scenarios to ensure balanced evaluation. The results demonstrated that the system successfully detected all spoofing attacks while maintaining stability during normal operation.

- True Positive Rate: 100% (all spoofing scenarios detected)
- False Positive Rate: 0% (no normal behavior incorrectly flagged)
- Detection Latency: < 50 ms per event
- Devices Simulated: 8
- Total Events Tested: ~85

These results indicate that the system performs accurately and efficiently under controlled testing conditions, with minimal delay and no incorrect classifications. The system maintained consistent performance due to its in-memory processing approach, which avoids database queries during the detection phase. This design significantly reduces latency and allows rapid evaluation of incoming telemetry. The use of a lightweight database (SQLite) did not impact performance under the tested workload, although it may introduce limitations under higher concurrency in future deployments.

The dashboard was also evaluated for responsiveness, with updates occurring through periodic polling of API endpoints. Alerts were displayed in near real time, and the system remained stable throughout all simulation runs without crashes or data inconsistencies. No critical defects were identified during testing. All detection methods behaved as expected, and no failures occurred in event processing, storage, or visualization. Minor improvements were noted in areas such as UI responsiveness and potential scalability concerns, but these did not impact the overall functionality of the system.

7. Security, Privacy, Accessibility & Compliance (O5)

The SecureGPS system was designed with a strong focus on security, data integrity, and ethical considerations, particularly given the sensitive nature of GPS-based systems. Since the core purpose of the project is to detect malicious behavior, security principles were incorporated directly into the system's architecture and implementation.

From a security perspective, the system ensures data integrity through the use of HMAC-SHA256 cryptographic signing. Each GPS event is signed at the time of ingestion using a secret key, allowing the system to verify whether any stored data has been modified after processing. This provides a forensic layer to the system, ensuring that alerts and event records can be trusted during analysis. Additionally, all input data is validated using Pydantic models, which helps prevent malformed or unexpected data from entering the system. The system also follows a server-side detection approach, meaning that it does not rely on trusting the GPS device itself. Instead,

all telemetry is treated as potentially untrusted input and is evaluated using multiple independent detection methods. This reduces the risk of attackers bypassing detection by manipulating a single parameter and strengthens the overall reliability of the system.

From a privacy standpoint, the system does not collect or store personally identifiable information (PII). All data processed is limited to device identifiers and location telemetry, which are used strictly for detection and monitoring purposes. This minimizes privacy risks and ensures that the system remains focused on security functionality rather than user tracking.

With ethical considerations taken into account during development, real-world GPS spoofing data was not used due to legal and regulatory restrictions, as generating or capturing spoofed signals can interfere with legitimate navigation systems. Instead, all testing was conducted using simulated data to ensure safe and controlled evaluation. The system is intended for defensive use only, supporting detection and prevention of spoofing attacks rather than enabling offensive capabilities.

Accessibility was addressed through a simple and intuitive dashboard design. The interface uses clear visual indicators, such as color-coded markers and alert levels, to make it easy for users to quickly interpret system status. The lightweight frontend ensures that the application can be accessed through standard web browsers without requiring specialized software.

From a compliance perspective, the system aligns with general cybersecurity best practices, including secure data handling, input validation, and integrity verification. While the current implementation does not include authentication or role-based access control due to its MVP scope, these features are identified as necessary additions for future production deployment.

8. Deployment & Operations

The SecureGPS system is deployed as a local application designed for development and demonstration. The backend runs using FastAPI and is served through the Uvicorn server, allowing it to process GPS telemetry and return API responses in real time. The system can be started with a simple command and accessed through a browser at <http://localhost:8000>, where users can view the live dashboard.

The dashboard retrieves data by periodically polling API endpoints, enabling real-time visualization of device activity, alerts, and system statistics. The system is lightweight and requires minimal setup, with the SQLite database automatically created and updated as events are processed. A reset endpoint is also available to clear data and restart the system for testing purposes. Additionally, the simulation script can be used to generate test data and demonstrate spoofing detection in a controlled environment.

While currently running locally, the system is designed to support future deployment in cloud environments using containerization and scalable services. Enhancements such as a production database, authentication, and real-time communication can be added to support larger-scale use. Detailed setup, usage, and operational procedures are provided in Appendix A (Installation Guide), Appendix B (User Manual), and Appendix C (Admin/Operations Guide).

9. Limitations & Future Work

While SecureGPS demonstrates strong performance in detecting GPS spoofing attacks, there are several limitations associated with the current implementation. The system was designed with simplicity and functionality in mind, which resulted in certain tradeoffs that impact scalability, security, and real-world applicability.

One of the primary limitations is the use of SQLite as the database, which does not scale well under high concurrency or large data volumes. Additionally, the system currently runs in a local environment and does not include cloud deployment, limiting its ability to support distributed or production-level use cases. The real-time dashboard relies on polling rather than more efficient communication methods such as WebSockets, which can introduce unnecessary overhead and latency in larger systems.

From a security standpoint, the system does not currently implement authentication or role-based access control, meaning that access to the dashboard and API is unrestricted in its current form. While this was acceptable for an MVP focused on detection capabilities, it would need to be addressed before deployment in a real-world environment.

Future improvements to the system focus on scalability, security, and enhanced detection capabilities:

- Cloud Deployment and Containerization: Deploy the system using Docker and cloud platforms (AWS, Azure, or GCP) to support scalability and real-world usage.
- Database Upgrade: Replace SQLite with a scalable database such as PostgreSQL to handle higher data volumes and concurrent users.
- Authentication and Access Control: Implement user authentication (e.g., JWT or OAuth) and role-based access control to secure system access.
- Real-Time Communication Enhancements: Replace polling with WebSockets to provide more efficient, low-latency updates to the dashboard.
- Advanced Detection Methods: Incorporate machine learning or adaptive models to improve detection accuracy and reduce reliance on fixed thresholds.
- Enhanced Geofencing: Support polygon-based geofences and multiple regions for more flexible and accurate boundary detection.
- Scalability and Performance Optimization: Improve system performance under high load and optimize data processing for larger-scale deployments.

10. References

- [1] FastAPI, "FastAPI Documentation," 2026. [Online]. Available: <https://fastapi.tiangolo.com/>
- [2] Leaflet.js, "Leaflet — Interactive Maps," 2026. [Online]. Available: <https://leafletjs.com/>
- [3] Python Software Foundation, "hashlib — Secure hashes and message digests," 2026. [Online]. Available: <https://docs.python.org/3/library/hashlib.html>
- [4] OWASP Foundation, "Input Validation Cheat Sheet," 2026. [Online]. Available: <https://owasp.org/www-project-cheat-sheets/>
- [5] GPS.gov, "GPS Accuracy," 2023. [Online]. Available: <https://www.gps.gov/>

[6] OpenStreetMap Contributors, “OpenStreetMap,” 2026. [Online]. Available: <https://www.openstreetmap.org/>

Appendices (Evidence & How-To)

A. Installation Guide (step-by-step with screenshots)

Step 1: Clone the Repository

```
git clone https://github.com/dbikowicz/GPSSpoofDetector.git
cd GPSSpoofDetector
```

Step 2: Create and Activate Virtual Environment

Mac/Linux

```
python3 -m venv .venv
source .venv/bin/activate
```

Windows (PowerShell)

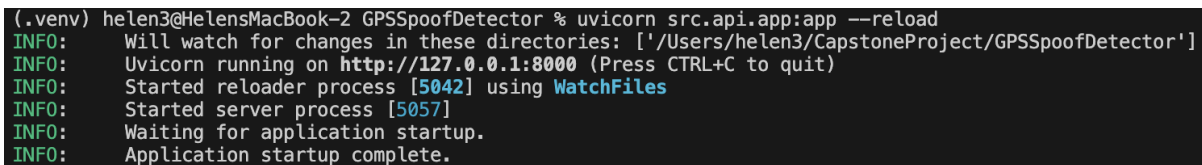
```
py -m venv .venv
.\.venv\Scripts\Activate.ps1
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

Step 4: Start the Server

```
uvicorn src.api.app:app --reload
```

A terminal window screenshot showing the command prompt and output for starting the SecureGPS server. The prompt is (.venv) helen3@HelensMacBook-2 GPSSpoofDetector % and the command is uvicorn src.api.app:app --reload. The output shows several INFO messages: 'Will watch for changes in these directories: [...] /Users/helen3/CapstoneProject/GPSSpoofDetector', 'Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)', 'Started reloader process [5042] using WatchFiles', 'Started server process [5057]', 'Waiting for application startup.', and 'Application startup complete.'

```
(.venv) helen3@HelensMacBook-2 GPSSpoofDetector % uvicorn src.api.app:app --reload
INFO: Will watch for changes in these directories: ['/Users/helen3/CapstoneProject/GPSSpoofDetector']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [5042] using WatchFiles
INFO: Started server process [5057]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Figure A1. SecureGPS server running locally using Uvicorn.

Step 5: Open the Application

Open your browser and go to: `http://localhost:8000`

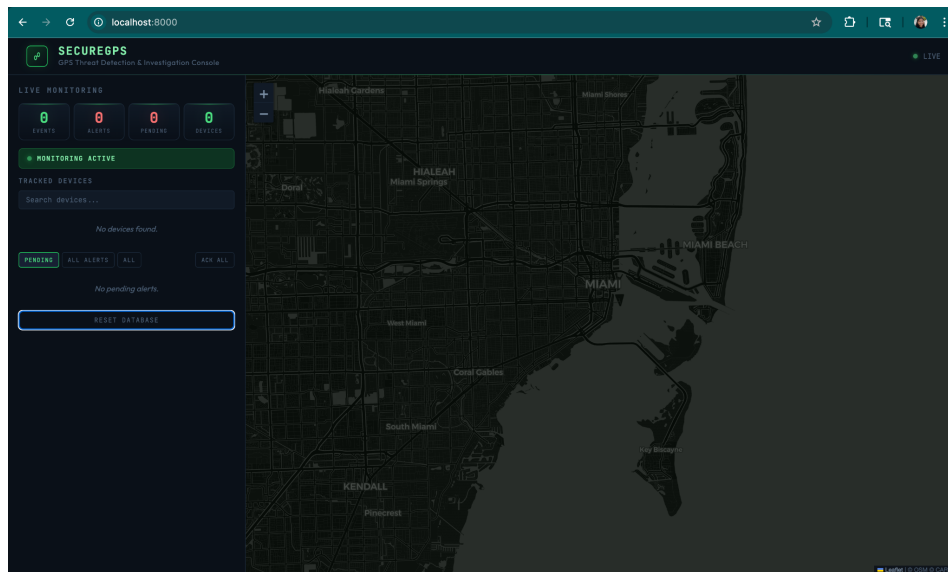


Figure A2. SecureGPS live dashboard running locally.

Step 6: Run Simulation (Optional)

```
python src/demo/send_scenarios.py
```

B. User Manual (task-oriented walkthroughs)

The SecureGPS dashboard provides a real-time interface for monitoring GPS telemetry and detecting spoofing events.

Viewing Device Activity:

- Devices are displayed as markers on the map, representing their real-time GPS location
- Green markers indicate normal activity where no spoofing behavior is detected
- Red markers indicate suspicious or potentially spoofed activity that has been flagged by the detection engine

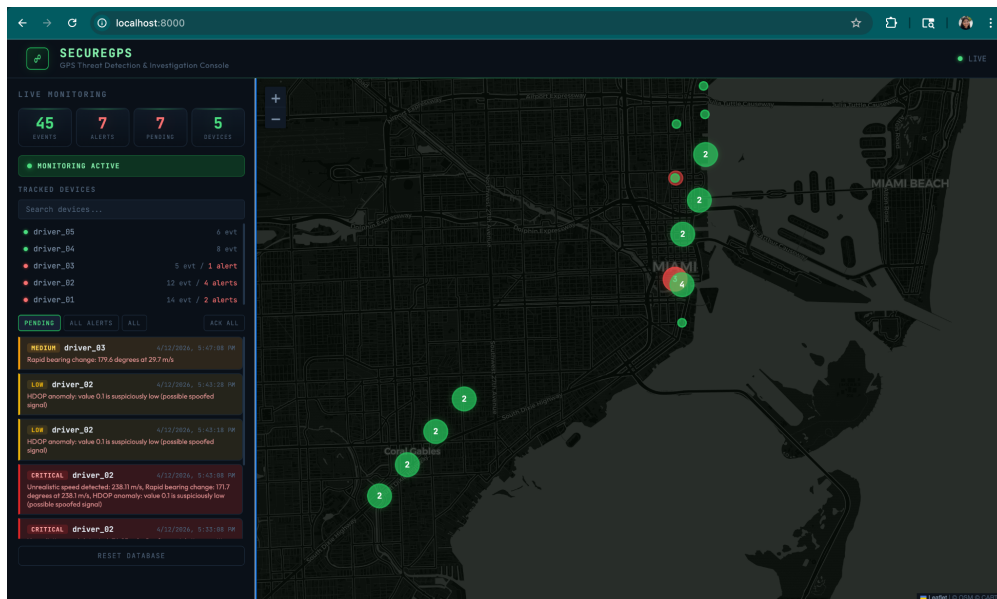


Figure B1. SecureGPS dashboard overview showing real-time device monitoring.

Using the Alert Feed

- Alerts are displayed in the sidebar, providing a list of detected events with associated severity levels
- Each alert includes details such as device ID, timestamp, and detection reason
- Users can filter alerts to focus on specific categories:
 - Pending – alerts that have not yet been reviewed
 - Acknowledged – alerts that have already been reviewed
 - All alerts – displays all detected events regardless of status

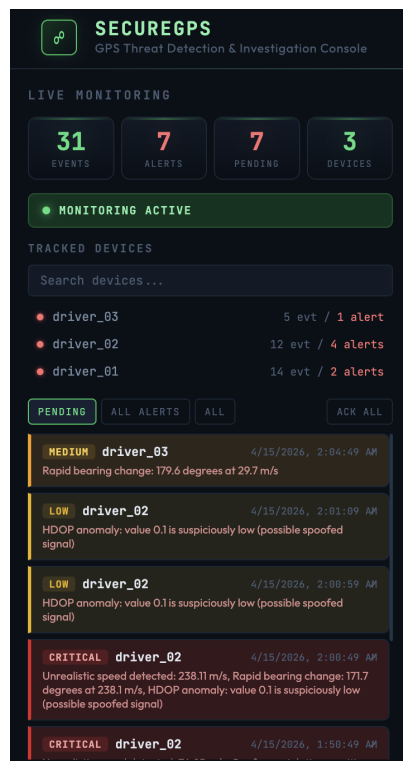


Figure B2. Alert feed displaying detected spoofing events with severity levels.

Interacting with Alerts

- Click on an alert to view more detailed information about the detected event
- Alerts can be acknowledged after review to indicate they have been investigated
- The acknowledgment feature supports a workflow similar to a security operations center (SOC), helping track reviewed and unresolved alerts

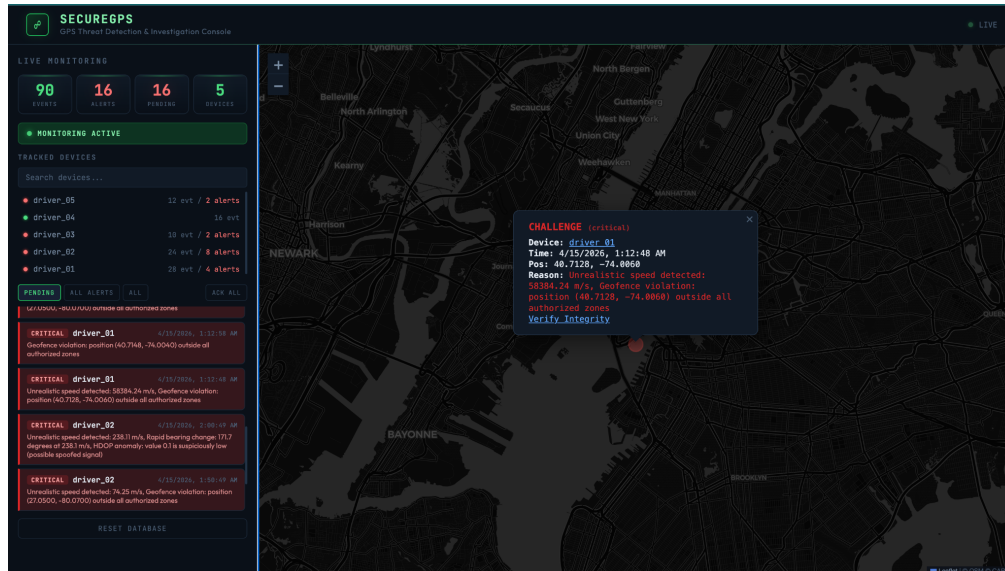


Figure B3. Example of a flagged spoofing event highlighted on the map.

Monitoring System Metrics

- View key metrics such as total events processed, number of alerts, and active devices
- Metrics update continuously, providing real-time visibility into system activity
- Helps users quickly assess system status and identify unusual patterns or spikes

C. Admin/Operations Guide (runbook, on-call, backup/restore)

The SecureGPS system includes basic administrative and operational controls for managing data and system behavior.

Resetting the System:

- To clear all stored events and reset system state:

```
curl -X POST http://localhost:8000/reset
```

Verifying Event Integrity:

- To verify if an event has been modified:

```
GET /verify/{id}
```

Monitoring System Health:

GET /health

Managing Data

- Events are stored in securegps.db
- Data can be cleared or reset for testing purposes

D. API Reference (OpenAPI/GraphQL schema)

The SecureGPS system exposes REST API endpoints for data ingestion and retrieval.

Endpoint	Method	Description
/ingest	POST	Submit GPS data
/events	GET	Retrieve events
/alerts	GET	Retrieve alerts
/stats	GET	System stats
/verify/{id}	GET	Verify integrity
/health	GET	Check system

Full API documentation available at /docs when the server is running.

E. Poster (36"×48" horizontal), Slides, and Video Links (Intro, Demo)

The following materials were created as part of the SecureGPS capstone project:

- Poster (36"×48" horizontal): Provides an overview of the system architecture, detection methods, and evaluation results
- Presentation Slides : Used to present the project, including problem context, system design, and demonstration
- Intro Video : Brief overview of the project, goals, and key features
- Demo Video: Demonstrates system functionality, including real-time spoofing detection and dashboard interaction

Access to these materials is available at:

Poster:  SecureGPS_Poster 2.pdf

Slides:  SecureGPS_Capstone_Presentation.pdf

Intro Video:  FIU-SCIS-2026Spring-CyberSecurity-SecureGPS-IntroVideo

Demo Video:  FIU-SCIS-2026Spring-CyberSecurity-SecureGPS-UserGuide

F. Scrum Evidence Index (links to Sprint Planning, Dailies, Reviews, Retros)

The following artifacts document the Agile development process used throughout the SecureGPS project:

- Sprint backlog grooming meeting notes
- User stories and prioritized backlog items
- Task tracking and development progress

These artifacts are available at:  Documents

4. Discipline-Specific Depth (O6)

The SecureGPS system was designed under the assumption that all incoming GPS telemetry is untrusted and potentially malicious. The primary asset being protected is the integrity and reliability of GPS location data, as well as the accuracy of detection results and stored event records.

The main adversary considered is an attacker using low-cost GPS spoofing tools, such as software-defined radios (SDRs), to manipulate location data and mislead tracking systems. These attackers may attempt to inject false coordinates, simulate unrealistic movement, or evade detection by mimicking normal behavior.

The system's attack surface includes the telemetry ingestion endpoint (/ingest), which receives external data, as well as the database where events are stored and the dashboard used for monitoring. If not properly secured, these components could be targeted for data injection, tampering, or unauthorized access.

A simplified STRIDE-based analysis was applied:

- Spoofing: Attackers sending falsified GPS data (core threat addressed by the system)
- Tampering: Modification of stored event data (mitigated through HMAC signing)
- Repudiation: Lack of traceability (addressed through event logging and signatures)
- Information Disclosure: Limited risk due to no storage of sensitive user data
- Denial of Service: Potential risk through excessive telemetry requests
- Elevation of Privilege: Not currently applicable due to lack of authentication (identified as future work)

Several security controls were implemented to mitigate identified risks. Input validation is enforced using Pydantic models to ensure all incoming telemetry data follows a strict schema, reducing the risk of malformed or malicious input. Cryptographic integrity is achieved through HMAC-SHA256 signing of each event, allowing the system to detect any post-ingestion tampering.

The system also uses a server-side detection model, meaning that GPS devices are not trusted and all data is verified independently. Logging and event tracking provide visibility into system activity and support auditability during analysis. While authentication and authorization are not currently implemented, they are identified as necessary future enhancements. The system design aligns with general secure coding practices and industry guidelines, including input validation and data integrity principles commonly referenced in OWASP.

Security testing was conducted through controlled simulation of spoofing scenarios rather than traditional SAST or DAST tools. A custom simulation script was used to generate both normal and malicious GPS data, allowing validation of detection methods under realistic conditions.

Testing confirmed that the system successfully detected all simulated spoofing attacks, including teleportation, geofence violations, HDOP anomalies, and bearing reversals, with a positive rate during normal operation. Additionally, event integrity verification was tested by recomputing HMAC signatures to confirm that stored data had not been altered. Dependency-level security was indirectly addressed by using standard Python libraries and a controlled environment with defined dependencies (requirements.txt). No known critical vulnerabilities were identified during development, although formal SAST/DAST and SBOM scanning are recommended as future improvements.