Project Plan: AI-Powered Intrusion Detection System

Portfolio Project for Kamal Dabban

May 22, 2025

Prepared for: Kamal Dabban, Third-Year Computer Engineering Student

Toronto Metropolitan University

Contents

1 Objective

Develop a web-based Network Intrusion Detection System (NIDS) that connects to a PostgreSQL database of network logs, uses AI to detect anomalies, and visualizes results via a dashboard. The project showcases Python, SQL, machine learning, and data visualization skills for cybersecurity roles.

2 Scope

- **Data**: NSL-KDD or CICIDS2017 datasets (timestamp, IP, protocol, packet size, labels).
- **AI**: Isolation Forest for anomaly detection.
- **Database**: PostgreSQL for log storage and querying.
- **Features**: Dashboard with anomaly visualizations, alerts, role-based access control.
- **Deliverables**: Web app, GitHub repository, technical report with UML diagrams, live demo.

3 Project Phases and Timeline (8–12 Weeks)

3.1 Phase 1: Setup and Data Acquisition (Weeks 1–2)

- **Tasks**:
  - Install Python, PostgreSQL, pandas, scikit-learn, Flask, Chart.js.
  - Download NSL-KDD dataset from Kaggle or CICIDS2017 from Canadian Institute for Cybersecurity.
  - Design PostgreSQL schema (e.g., `logs: id, timestamp, src_ip, dst_ip, protocol, packet_size, label`) and load data.
- **Milestones**: Python environment configured, dataset loaded, basic SQL queries.
- **Tools**: Python, PostgreSQL, pandas, Git.

3.2 Phase 2: Data Preprocessing and Model Development (Weeks 3–5)

- **Tasks**:
  - Clean NSL-KDD data with pandas (handle missing values, encode categorical features).
  - Train Isolation Forest model (scikit-learn) for anomaly detection; evaluate with precision, recall, F1-score.
  - Integrate SQL queries for data retrieval (e.g., `SELECT * FROM logs WHERE timestamp > '2025-01-01'`).
- **Milestones**: Cleaned dataset, trained model with >80% F1-score, SQL integration.

- **Tools**: Python (pandas, scikit-learn), PostgreSQL.

3.3 Phase 3: Web Application Development (Weeks 6–8)

- **Tasks**:
  - Build Flask API for model predictions and database queries.
  - Develop dashboard with HTML/CSS and Chart.js for anomaly visualization (scatter chart).
  - Implement role-based access control with Flask-Login.
- **Milestones**: Functional API, dashboard with scatter chart, authentication system.
- **Tools**: Flask, HTML/CSS, Chart.js, Flask-Login.

3.4 Phase 4: Testing, Documentation, and Deployment (Weeks 9–12)

- **Tasks**:
  - Test model and web app with pytest; validate functionality.
  - Document system with UML diagrams and technical report.
  - Deploy on Heroku or AWS with public URL.
- **Milestones**: Tested app, completed documentation, live demo.
- **Tools**: pytest, GitHub, Heroku/AWS, LaTeX.

4 Resources Needed

- **Software**: Python 3.9+, PostgreSQL, Flask, scikit-learn, Chart.js, Git, Heroku/AWS CLI.
- **Hardware**: Laptop with 8GB+ RAM.
- **Datasets**: NSL-KDD (500MB), CICIDS2017 (2GB).
- **Learning Resources**: Scikit-learn documentation, Flask tutorials, Chart.js examples, PostgreSQL guides.

5 Risks and Mitigation

- **Large dataset slows preprocessing**: Use pandas; sample NSL-KDD for initial testing.
- **Low model accuracy**: Tune Isolation Forest hyperparameters; compare with Autoencoder.
- **Deployment issues**: Follow Herokus Flask guide; test locally.

6 Success Criteria

- Web app detecting anomalies with >80% F1-score.
- Dashboard with interactive scatter chart.
- Secure database with role-based access.
- GitHub repository with documentation and live demo.
- Technical report with UML diagrams.