### **Project Report: CyScan - Real-time** APT **Detection System**

### **1. Executive Summary**

Project CyScan has successfully achieved its initial objective: to design, build, and deploy a fully functional, end-to-end data pipeline for real-time host monitoring and anomaly detection. The system is capable of collecting system-level process data from the host, analyzing it in real-time with a machine learning model, and storing generated alerts in a scalable database for visualization. The current implementation serves as a robust and scalable foundation for further development in advanced threat detection.

### **2. System Architecture**

The architecture of CyScan is modeled after modern, event-driven security data platforms, emphasizing scalability, resilience, and modularity. The system consists of several key components that form a logical data flow from collection to visualization.

Data Flow:

Host Data -> Python Producer -> Kafka Message Bus -> Faust Detection Engine -> Kafka Message Bus -> Python Alert Sink -> Elasticsearch Database -> Kibana Dashboard

**Key Architectural Components:**

* **Containerized Infrastructure (docker-compose.yml):** The entire backend infrastructure is containerized using Docker. This includes Apache Kafka for message brokering, Elasticsearch for data storage, and Kibana for visualization. This approach ensures a consistent, reproducible, and isolated environment for the core services.
* **Decoupled Microservices:** Each logical function of the pipeline (producing, detecting, sinking) is handled by a separate, independent Python script. These scripts communicate exclusively through Kafka topics, a design that makes the system highly scalable and easy to maintain.

### **3. Component Implementation**

Based on the project files, the following components have been implemented and are fully operational:

* **Data Producer (producer.py):**
  + **Function:** This script acts as the primary data collector. It uses the psutil library to gather a snapshot of all running processes on the host machine every five seconds.
  + **Data Logging:** In addition to its real-time function, the producer logs the numerical features (pid, ppid, create\_time) of these processes to a baseline\_data.csv file. This file serves as the training dataset for the machine learning model.
  + **Kafka Integration:** The script serializes the collected process data into JSON format and publishes each event as a message to the osquery-events Kafka topic.
* **Machine Learning Model (train\_model.py, isolation\_forest\_model.joblib):**
  + **Function:** An offline training pipeline has been established to create an anomaly detection model. The train\_model.py script reads the baseline\_data.csv, prepares the data using pandas, and trains an **Isolation Forest** model using the scikit-learn library.
  + **Output:** The successfully trained and ready-to-use model is saved to the isolation\_forest\_model.joblib file.
* **Detection Engine (detection\_engine.py):**
  + **Function:** This is the analytical core of the system. Built using the faust-streaming library, it is a real-time stream processor.
  + **ML Integration:** Upon startup, the engine loads the pre-trained isolation\_forest\_model.joblib.
  + **Real-time Inference:** It consumes process events from the osquery-events Kafka topic. For each event, it calculates an anomaly score using the loaded model. If the score exceeds a predefined threshold (-0.15), it flags the event as an anomaly.
  + **Alert Generation:** When an anomaly is detected, the engine generates a new, structured JSON alert and publishes it to a separate security-alerts Kafka topic.
* **Alert Sink (alert\_sink.py):**
  + **Function:** This script serves as the bridge between the real-time pipeline and long-term storage.
  + **Data Persistence:** It consumes alert messages from the security-alerts topic and uses the elasticsearch Python library to index each alert as a document in the cyscan-alerts index within Elasticsearch. This ensures all generated alerts are durably stored.

### **4. Current Status & Achievements**

* **End-to-End Pipeline:** The project is fully functional from data collection to alert visualization.
* **Real-time Processing:** The use of Kafka and Faust enables the system to process system events with low latency.
* **ML-Powered Detection:** The system has successfully integrated a machine learning model (Isolation Forest) for anomaly detection, moving beyond simple, static rules.
* **Scalable Foundation:** The underlying Docker and Kafka architecture is highly scalable and can serve as a foundation for monitoring multiple endpoints or incorporating more complex detection logic in the future.

The project has successfully met all its initial goals and stands as a comprehensive, working prototype of a modern security monitoring system.