Project Report: Real-Time AI-Powered IDS

1. Introduction

This document serves as a comprehensive report and outlines the business and functional requirements for the Real-Time AI-Powered Intrusion Detection System (IDS). The project aims to develop a robust system capable of monitoring network traffic in real-time, leveraging artificial intelligence and machine learning techniques to detect anomalies, cyber threats, and various malicious activities. Key activities include Distributed Denial of Service (DDoS) attacks, malware communications, and unauthorized access attempts. The system integrates several open-source technologies, including Suricata for network traffic analysis, nDPI for deep packet inspection, an Isolation Forest model for anomaly detection, and the ELK (Elasticsearch, Logstash, Kibana) stack for logging, analysis, and visualization. All components are designed to be containerized using Docker for ease of deployment, reproducibility, and scalability.

2.Table of Context

3. Project Objectives

The primary objectives of this project, as confirmed by the user, are:

- **Build a real-time Intrusion Detection System (IDS):** Develop a system capable of continuous network traffic monitoring and immediate threat identification.
- **Use Machine Learning (ML) to detect malicious activities:** Implement AI-driven anomaly detection to identify suspicious patterns that may indicate attacks, going beyond traditional signature-based methods.
- **Analyze packet captures and log files:** Provide capabilities to process and analyze both historical packet capture (PCAP) files for model training and investigation, and real-time log files for ongoing monitoring and alert generation.

4. Scope

4.1. In Scope

- **Real-time Traffic Monitoring:** Integration of Suricata to capture and inspect network packets from a live interface (e.g., `eth0`).
- **Signature-Based Detection:** Utilization of Suricata with standard (e.g., Emerging Threats) and custom rule sets for known threat patterns.
- **Deep Packet Inspection (DPI) for Feature Extraction:** Conceptual use of nDPI to extract detailed features from network flows, primarily for training the ML model. (Note: `ndpiReader` for CSV generation from PCAPs is an external step as per the plan).
- **Machine Learning Model for Anomaly Detection:**
    - Training an Isolation Forest model using Python (scikit-learn) on benign network traffic data.

    - Incorporation of derived features (e.g., packet rate, byte rate) to enhance detection.
    - Tuning of the anomaly detection threshold using validation datasets to optimize performance (e.g., F1-score).
- **Real-time Anomaly Scoring:** A Python script to process Suricata EVE JSON flow logs, apply the trained ML model, and score flows for anomalies.
- **Log Management and Storage:** Use of the ELK stack (Elasticsearch, Filebeat, Kibana version 9.0.1) for collecting, storing, and indexing Suricata logs and detected anomalies.
- **Visualization:** Configuration of Kibana dashboards for viewing Suricata alerts, network flows, and a dedicated dashboard for AI-detected anomalies.
- **Containerization:** Packaging of all core components (Suricata, Elasticsearch, Kibana, Filebeat) using Docker and Docker Compose for simplified deployment and management.
- **Documentation:** Creation of a detailed project report (this document) and a comprehensive README file with setup and usage instructions.
- **Repository:** Provision of a well-structured GitHub repository containing all code, configurations, and documentation.

4.2. Out of Scope

- Automated incident response beyond alerting and logging.
- Development of a graphical user interface (GUI) beyond Kibana dashboards.
- Commercial IDS features or proprietary software integration unless explicitly stated (e.g., specific commercial rule sets).
- Guaranteed detection of all zero-day attacks (ML models are based on learned patterns).
- Performance benchmarking for specific high-throughput enterprise environments beyond the planned testing simulations.
- The direct inclusion of `ndpiReader` or other PCAP-to-CSV conversion tools within the primary Docker build of Suricata; this is treated as a pre-requisite step for generating training data.

6. System Architecture and Components

The IDS architecture is a multi-stage pipeline:

1. **Network Tap/PCAP Input:** Suricata ingests network traffic either from a live interface or PCAP files.
2. **Traffic Inspection & Rule Matching (Suricata):** Suricata inspects traffic using its rule engine (including custom rules for SYN floods, port scans) and nDPI for protocol identification. It generates EVE JSON logs containing alerts, flow data, HTTP logs, DNS logs, etc.
3. **Log Aggregation (Filebeat):** Filebeat monitors Suricata's EVE JSON output file and ships new log entries to Elasticsearch.
4. **Data Storage & Indexing (Elasticsearch):** Elasticsearch stores and indexes the logs, making them searchable and analyzable.
5. **Data Visualization & Exploration (Kibana):** Kibana connects to Elasticsearch, providing tools to query data and visualize it through dashboards. This includes default Suricata dashboards and a custom dashboard for ML-detected anomalies.
6. **ML Model Training (Offline - Python, scikit-learn):**
    - PCAP files (e.g., CIC-IDS2017) are processed by `ndpiReader` (externally) to create CSV datasets of network flows with relevant features.
    - The `train_model.py` script uses these CSVs (benign and attack validation sets) to train an Isolation Forest model. It performs feature engineering (e.g., packet/byte rates) and tunes the anomaly decision threshold. The trained model is saved (e.g., `ids_model.pkl`).
7. **Real-Time Anomaly Detection (Online - Python, Elasticsearch, scikit-learn):**
    - The `detect_anomalies.py` script runs continuously.
    - It queries Elasticsearch for new Suricata flow records.

- It preprocesses these flows, calculating derived features consistent with the training phase.
- It applies the loaded Isolation Forest model to score each flow.
- Flows identified as anomalous (based on the tuned threshold) are enriched with an anomaly score and indexed into a separate Elasticsearch index (`suricata-anomalies`) for focused review in Kibana.

Key Technologies:

- **Suricata (v7.0.10):** Open-source IDS/IPS/NSM engine.
- **nDPI:** Open-source deep packet inspection library.
- **Python (v3.x):** For scripting ML model training and real-time detection.
    - **scikit-learn:** For the Isolation Forest algorithm and metrics.
    - **pandas, numpy:** For data manipulation.
    - **joblib:** For saving/loading the trained model.
    - **elasticsearch-py:** For interacting with Elasticsearch.
- **ELK Stack (v9.0.1 recommended in plan, using official images):**
    - **Elasticsearch:** Distributed RESTful search and analytics engine.
    - **Kibana:** Visualization and exploration tool for Elasticsearch.
    - **Filebeat:** Lightweight log shipper.
- **Docker & Docker Compose:** For containerization and orchestration.
- **Bash:** For system-level scripting if needed (primarily within Dockerfiles).

6. Implementation Details

This section details the configuration and code for each component as per the project plan.

6.1. Docker Environment

- **`docker/Dockerfile.suricata`:**
    - Base Image: `ubuntu:22.04`.
    - Installs build dependencies, Python3, pip, and `suricata-update`.
    - Clones, compiles, and installs nDPI from source.
    - Clones, compiles (with nDPI support), and installs Suricata 7.0.10 from source.
    - Runs `ldconfig` to update shared library links.
    - Uses `suricata-update` to fetch and enable the ET/Open rule set.
- **`docker/docker-compose.yml`:**
    - Defines services: `elasticsearch`, `kibana`, `filebeat`, `suricata`.
    - `elasticsearch`: Official image, single-node, basic security (`elastic`/`changeme`), persistent volume for data.
    - `kibana`: Official image, connects to Elasticsearch with credentials.
    - `filebeat`: Official image, mounts `configs/filebeat.yml` and `logs/` directory, depends on Elasticsearch and Kibana.
    - `suricata`: Builds from `Dockerfile.suricata` (context set to project root `..`). `cap_add: NET_ADMIN`, `network_mode: host`. Mounts `configs/suricata.yaml`, `configs/custom.rules`, `pcap/`, and `logs/`. Command: `suricata -c /etc/suricata/suricata.yaml -i eth0`.

6.2. Suricata Configuration

- **`configs/suricata.yaml`:**
    - `default-rule-path: /var/lib/suricata/rules`
    - `rule-files`: Includes `suricata.rules` (from `suricata-update`) and `/etc/suricata/rules/custom.rules`.
    - `plugins`: Loads nDPI plugin.

    - `outputs.eve-log`: Enabled, regular file type to `/var/log/suricata/eve.json`. Includes detailed types for `alert`, `flow` (with specified fields like `pkts_toserver`, `app_proto`), `http`, `dns`, `tls`, `files`, `ssh`, `smtp`.
    - `af-packet`: Configured for `eth0` interface.
- **`configs/custom.rules`**:
    - Includes example rules for detecting potential SYN Floods and Port Scans based on thresholds.
        - `alert tcp any any → any any (msg:"Possible SYN Flood"; flags:S; flow:to_server; threshold: type both, track by_src, count 100, seconds 1; sid:1000001; rev:1;)`
        - `alert tcp any any → any any (msg:"Possible Port Scan"; flags:S; flow:to_server; detection_filter: track by_dst, count 50, seconds 10; sid:1000002; rev:1;)`

6.3. ELK Stack Integration

- **`configs/filebeat.yml`**:
    - `filebeat.modules`: Enables the `suricata` module.
    - `eve.var.paths`: Points to `/var/log/suricata/eve.json`.
    - `output.elasticsearch`: Configured for `elasticsearch:9200` with credentials.
    - `setup.kibana`: Configured for `kibana:5601` with credentials to allow Filebeat to set up dashboards.

6.4. Machine Learning Model (`scripts/`)

- **`scripts/requirements.txt`**: Lists Python dependencies: `pandas`, `numpy`, `scikit-learn`, `joblib`, `elasticsearch`.
- **`scripts/train_model.py`**:
    - Loads benign training data (e.g., `pcap/flows_benign.csv`).
    - Preprocesses data: Replaces duration 0 with 1e-6 to avoid division by zero.
    - Feature Engineering: Calculates `packet_rate` and `byte_rate`.
    - Selects features: `duration`, `src2dst_packets`, `dst2src_packets`, `src2dst_bytes`, `dst2src_bytes`, `packet_rate`, `byte_rate`.
    - Trains an `IsolationForest` model (`contamination=0.01`, `random_state=42`).
    - Saves the model to `scripts/ids_model.pkl`.
    - If validation data (`pcap/flows_val_benign.csv`, `pcap/flows_attack.csv`) exists, it loads them, preprocesses, and concatenates.
    - Generates labels (0 for benign, 1 for attack) for the validation set.
    - Tunes the anomaly `threshold` by iterating through a range of values and selecting the one that maximizes the F1-score on the validation set.
    - Prints the best threshold and F1-score, advising the user to update `detect_anomalies.py`.
    - Writes the optimal threshold to `scripts/optimal_threshold.txt` for easier use by the detection script.
- **`scripts/detect_anomalies.py`**:
    - Connects to Elasticsearch.
    - Loads the trained `ids_model.pkl`.
    - Attempts to load the `optimal_threshold` from `scripts/optimal_threshold.txt`; uses a default if not found.
    - Enters a loop (polling every `QUERY_INTERVAL_SECONDS`):
        - Queries Elasticsearch for new Suricata flow records (`event.dataset: suricata.flow`) since the last query.
        - Preprocesses retrieved flows: Calculates duration, packet/byte rates, ensuring all features required by the model are present.
        - Applies the Isolation Forest model (`decision_function`) to get anomaly scores.
        - Flags flows with scores below `OPTIMAL_THRESHOLD` as anomalies.
        - Indexes detected anomalies (original flow data + anomaly score + detection timestamp) into the `suricata-anomalies` Elasticsearch index.
        - Includes error handling for Elasticsearch connection issues and model loading.

6.6. Kibana Dashboards

- Filebeat automatically sets up default Suricata dashboards.
- A placeholder for `dashboards/suricata_anomalies.ndjson` is created. This file would be generated by exporting a custom Kibana dashboard designed to visualize data from the `suricata-anomalies` index (e.g., tables of anomalies, time-series charts of anomaly counts, source/destination IP breakdowns for anomalies).

7. Usage Instructions / Deployment

(Detailed instructions will be in the README.md, summarized here)

1. **Prerequisites:** Linux (Ubuntu recommended), Docker, Docker Compose, Python3/pip for local script execution if needed, PCAP files for training.
2. **Clone Repository.**
3. **Prepare PCAP Data:** Place `train_benign.pcap`, `val_benign.pcap`, `attack.pcap` in `pcap/`. Use `ndpiReader` (external tool) to convert these to `flows_benign.csv`, `flows_val_benign.csv`, `flows_attack.csv` respectively, and place them in `pcap/`.
4. **Train Model:** Navigate to `scripts/` and run `python3 train_model.py`. Note the suggested `optimal_threshold` or ensure `optimal_threshold.txt` is created.
5. **Build and Start Services:** Navigate to `docker/` and run `sudo docker-compose up -d --build`.
6. **Setup Filebeat/Kibana:** After services are up, run:
    - `sudo docker exec <filebeat_container_id> filebeat modules enable suricata`
    - `sudo docker exec <filebeat_container_id> filebeat setup --dashboards`
    - `sudo docker-compose restart filebeat` (from `docker/` directory)
7. **Start Real-Time Detection:** Navigate to `scripts/` and run `python3 detect_anomalies.py`. This should run continuously.
8. **Monitor:** Access Kibana at `http://localhost:5601` (user: `elastic`, pass: `changeme`). Import `dashboards/suricata_anomalies.ndjson` if manually created and exported.

8. Assumptions

- The host system has Docker and Docker Compose correctly installed and operational.
- Sufficient system resources (CPU, RAM - 8GB+ recommended) are allocated to Docker.
- The network interface specified in `suricata.yaml` and `docker-compose.yml` (e.g., `eth0`) is correct for the deployment environment.
- PCAP files for training and validation are available and representative.
- The `ndpiReader` tool is available externally for converting PCAPs to the CSV format required by `train_model.py`.
- The features extracted by `ndpiReader` and those available in Suricata's EVE JSON flow logs are semantically consistent for the ML model.

9. Limitations & Potential Challenges

- **Docker Environment Dependency:** As noted in previous interactions, the sandbox environment had issues starting the Docker daemon due to `iptables/nftables` conflicts. A fully functional Docker environment is critical.
- **Model Generalization:** The Isolation Forest model's performance depends heavily on the training data. It may not detect novel attacks or sophisticated evasions.
- **False Positives/Negatives:** Anomaly detection systems are prone to false alarms. The `contamination` parameter and `optimal_threshold` require careful tuning and ongoing adjustment based on observed traffic and feedback.
- **Encrypted Traffic:** Visibility into encrypted traffic is limited to metadata (e.g., TLS handshake parameters, flow characteristics). The content of encrypted sessions cannot be inspected by nDPI or Suricata rules directly.
- **Performance at Scale:** While designed for real-time operation, extremely high network throughput might require further performance tuning of Suricata, Elasticsearch, and the Python detection script.
- **Resource Consumption:** Running multiple Docker containers (ELK, Suricata) can be resource-intensive.
- **Security:** Default credentials for Elasticsearch (`elastic` / `changeme`) are used and **must be changed** in any production or sensitive environment.

10. Testing Strategy (as per project plan)

- **Unit Testing:** Individual scripts (e.g., `train_model.py`, `detect_anomalies.py`) should be tested with sample data.
- **Integration Testing:** Verify the data flow: Suricata → Filebeat → Elasticsearch → Kibana, and Suricata → Elasticsearch → `detect_anomalies.py` → Elasticsearch (anomalies index).
- **Attack Simulation:** Use `tcpreplay` with various malicious PCAP files (e.g., SYN floods, port scans, malware C&C traffic) to test both Suricata rule-based detection and ML-based anomaly detection. Verify alerts and anomalies in Kibana.
- **Performance Testing (Basic):** Monitor CPU/memory usage of containers under simulated load.

11. Conclusion

This project provides a comprehensive plan and implementation for a Real-Time AI-Powered Intrusion Detection System. By integrating Suricata, nDPI concepts, an Isolation Forest model, and the ELK stack within a Dockerized environment, it meets the core objectives of real-time monitoring, ML-based threat detection, and log analysis. The system is designed to be extensible and provides a solid foundation for further enhancements in network security monitoring. Successful deployment and validation are contingent on a compatible host environment for Docker.