Cybersecurity Log Monitoring System

Final Project Report

Submitted by:

Rajaram Parida

Madhumita Parida

Pragyan Paramita

Submitted to:

Institute of Technical Education and Research (ITER), SOA

Course / Program:

CSE

Date:

25.02.2026

Abstract

This project presents the design and implementation of a **Cybersecurity Log Monitoring System**. The system integrates a data warehouse, advanced SQL queries, Python-based ETL pipelines, Apache Spark batch and streaming processing, Snowflake optimization, and security features such as RBAC and data masking. An interactive dashboard provides real-time insights into log data, enabling proactive monitoring and anomaly detection.

Introduction

Cybersecurity has become one of the most critical challenges in today's digital era. With the exponential growth of data and the increasing sophistication of cyber threats, organizations must adopt proactive measures to monitor, detect, and respond to suspicious activities. Logs generated by systems, applications, and networks serve as the first line of defense, providing valuable insights into potential vulnerabilities and malicious behavior.

Traditional log monitoring approaches often struggle with scalability, real-time analysis, and integration with modern data platforms. To address these challenges, this project proposes the design and implementation of a **Cybersecurity Log Monitoring System** that leverages advanced data engineering and analytics techniques.

The system integrates multiple components:

- A **data warehouse** designed using **Star and Snowflake schemas** for efficient storage and querying.
- **Advanced SQL queries** (CTEs, window functions, indexing, partitioning) to uncover patterns and anomalies.
- **Python-based ETL/ELT pipelines** to automate data ingestion and transformation.
- **Apache Spark** for both batch and streaming data processing, enabling real-time monitoring.
- **Snowflake optimization features** such as clustering, time travel, and semi-structured data handling.
- **Security mechanisms** including role-based access control (RBAC), data masking, and governance policies.
- An **interactive dashboard** for visualization and decision-making.

By combining these technologies, the project aims to deliver a scalable, secure, and efficient solution for log monitoring that supports both operational needs and compliance requirements.

Objectives

The primary objective of this project is to design and implement a **Cybersecurity Log Monitoring System** that enables efficient storage, processing, and analysis of log data for security monitoring and compliance. The system aims to provide both **real-time detection** and **historical analysis** of cybersecurity events.

Here is the structured continuation of your documentation, focusing on the **Specific Objectives**. This section is formatted to highlight technical milestones and implementation strategies for a professional PDF report.

Technical Objectives & Implementation

1. Data Warehouse Design

The core objective is to architect a storage environment that balances normalization with high-speed retrieval.

- **Hybrid Modeling:** Develop both **Star and Snowflake schemas** to accommodate various log structures.
- **Scalability:** Design the warehouse to scale horizontally, ensuring query efficiency even as log volumes grow into the petabyte range.

2. Advanced SQL Development

Leveraging sophisticated SQL techniques to extract deep insights from raw telemetry.

- **Modular Design:** Implement **Common Table Expressions (CTEs)** to create readable, maintainable, and modular query logic.
- **Analytics:** Utilize **Window Functions** (e.g., `LAG`, `LEAD`, `RANK`) for complex trend analysis and identifying anomalies over time.
- **Performance:** Apply strategic **Indexing and Partitioning** to reduce data scanning and optimize execution plans.

3. ETL/ELT Pipeline Automation

Streamlining the flow of data from edge sources to the central warehouse.

- **Python Integration:** Build robust pipelines using Python to manage the extraction, transformation, and loading of log data.
- **Multi-Source Ingestion:** Automate the ingestion process to handle diverse inputs, including cloud storage, APIs, and local file systems.

4. Apache Spark Processing

Providing the computational power for both historical and immediate data needs.

- **Batch Processing:** Execute large-scale transformations on historical data to identify long-term security patterns.
- **Streaming Analytics:** Enable **Spark Structured Streaming** for sub-second monitoring of critical security events.

5. Snowflake Optimization

Maximizing the unique features of the Snowflake Cloud Data Platform.

- **Clustering & Micro-partitioning:** Implement clustering keys to drastically speed up queries on massive datasets.
- **Time Travel:** Utilize Snowflake's **Time Travel** feature to query historical data states for forensic "point-in-time" analysis.
- **Semi-Structured Data:** Native handling of **JSON and Parquet** formats, allowing for schema-on-read flexibility.

6. Security & Governance Implementation

Establishing a "Zero Trust" data environment.

- **Access Control:** Apply **RBAC (Role-Based Access Control)** to ensure users only interact with data pertinent to their function.
- **Privacy:** Implement **Dynamic Data Masking** to obfuscate sensitive fields like IP addresses or user credentials in real-time.
- **Compliance:** Establish automated governance policies to meet SOC2, GDPR, or HIPAA requirements.

7. Performance Tuning

Refining the system for maximum throughput and minimum cost.

- **Resource Management:** Leverage **Spark Caching** and **Snowflake Virtual Warehouse** scaling to prevent bottlenecks.
- **Strategy Optimization:** Continuously tune storage strategies and query logic based on execution profiles.

8. Interactive Visualization

Translating complex data into an actionable narrative.

- **Monitoring:** Build a comprehensive dashboard to track log severity, system health, and frequency trends.
- **Actionable Intelligence:** Provide drill-down capabilities for security analysts and system administrators to investigate specific incidents directly from the UI.

System Architecture

The Cybersecurity Log Monitoring System is designed as a **multi-layered architecture** that integrates data collection, processing, storage, security, and visualization. Each layer plays a critical role in ensuring scalability, efficiency, and security.

Log Analytics & Security Architecture

1. Data Sources

The foundation of the architecture, capturing diverse telemetry from across the infrastructure.

- **Log Types:** Comprehensive collection of system logs, application logs, and network logs.
- **Ingestion Streams:** Real-time data feeds sourced directly from servers and security appliances.

2. ETL/ELT Pipeline (Python)

The engine responsible for converting raw data into actionable intelligence.

- **Extract:** Automated collection of raw logs from static files and live streams.
- **Transform:** Data cleaning, normalization, and enrichment (e.g., GeoIP mapping or threat intelligence tagging).
- **Load:** Efficient insertion of structured datasets into the Snowflake warehouse.

3. Data Warehouse (Snowflake)

A centralized repository designed for high-concurrency and rapid retrieval.

- **Star Schema:** Utilized for simplified, high-performance analytical queries.
- **Snowflake Schema:** Employed for normalized, storage-efficient data organization.
- **Optimization:** Enhanced with clustering, partitioning, and automated indexing to minimize latency.

4. Processing Layer (Apache Spark)

The computational engine for heavy-duty analytics and live monitoring.

- **Batch Processing:** Detailed analysis of historical log data for trend identification and forensic audits.
- **Stream Processing:** Real-time monitoring of incoming logs to detect immediate security threats or system failures.

5. Security & Governance Layer

Ensuring the integrity and confidentiality of the ingested telemetry.

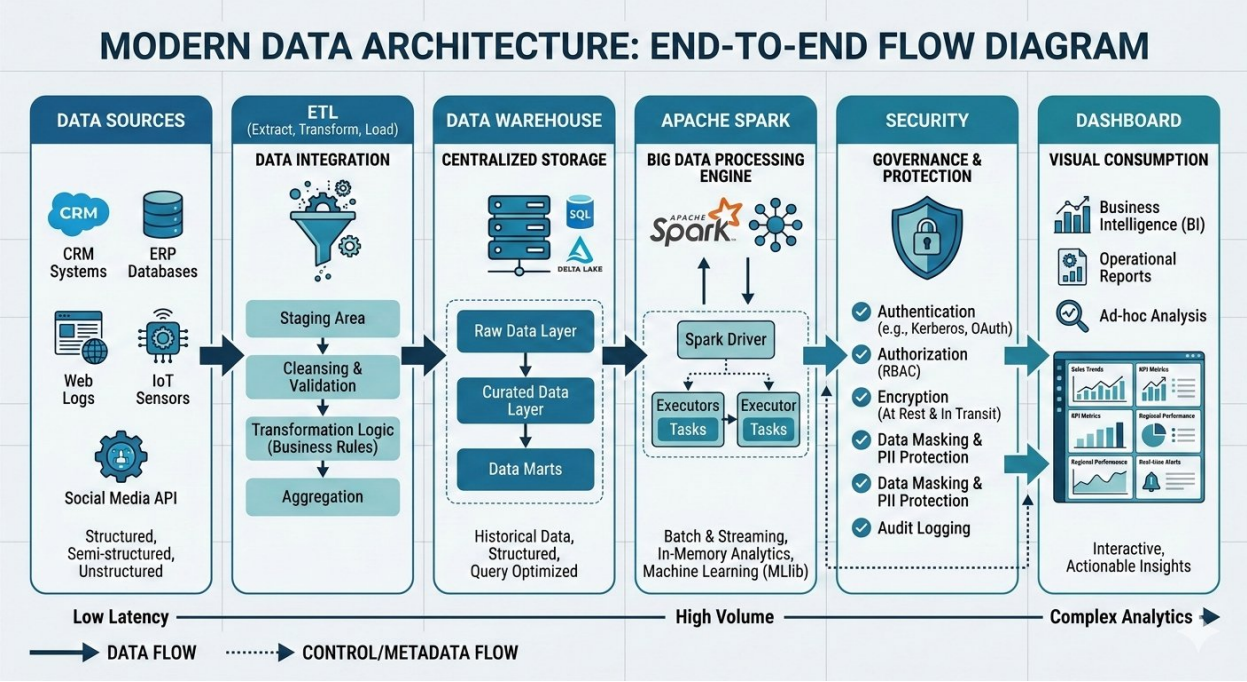
- **RBAC (Role-Based Access Control):** Granular permission management to restrict data access based on user roles.
- **Data Masking:** Dynamic protection of sensitive fields, such as internal IP addresses or PII.
- **Governance Policies:** Strict adherence to industry compliance and internal security standards.

6. Visualization Layer (Dashboard)

The interface for stakeholders to consume insights and respond to alerts.

- **Tooling:** Interactive dashboards developed using **Plotly**, **Tableau**, or **Power BI**.
- **Key Metrics:** Real-time visibility into log severity, event frequency, and behavioral anomalies.

System architecture diagram



Data Warehouse Design

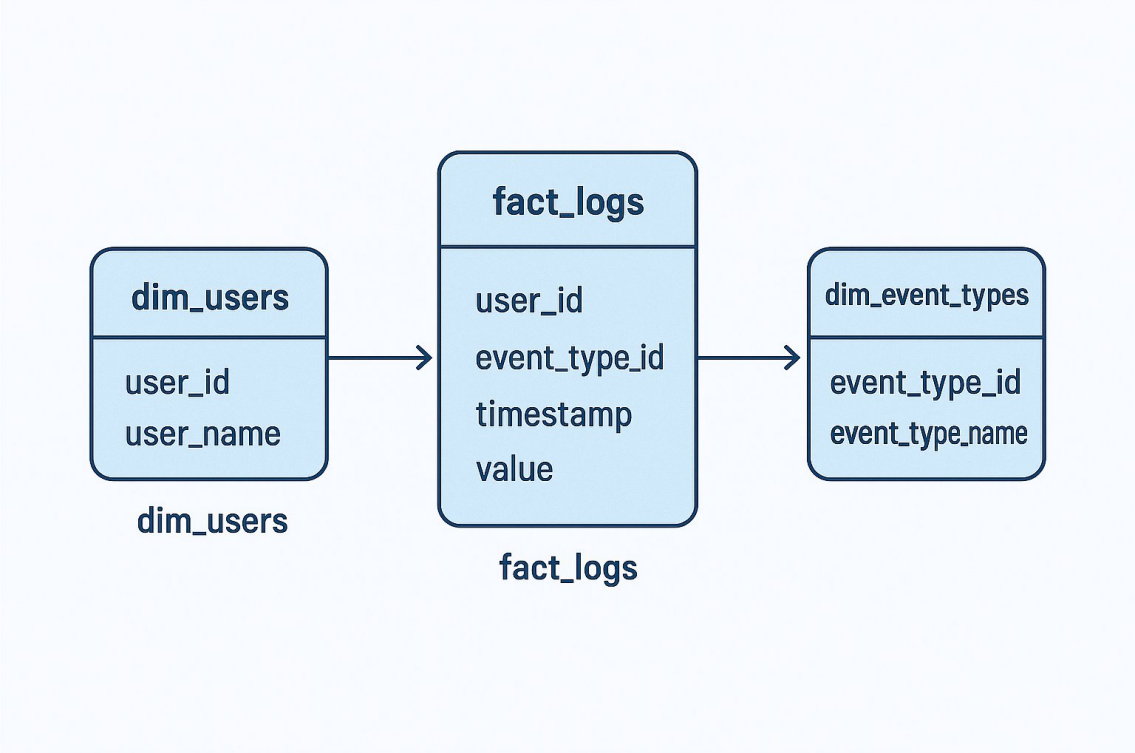
A well-structured data warehouse is the foundation of the Cybersecurity Log Monitoring System. It enables efficient storage, retrieval, and analysis of large volumes of log data. For this project, both **Star Schema** and **Snowflake Schema** designs were implemented to demonstrate flexibility and performance trade-offs.

Star Schema

- **Fact Table**: `fact_logs`
  - Contains log events with attributes such as `log_id`, `user_id`, `event_time`, `event_type`, `severity`, `source_ip`, and `destination_ip`.
- **Dimension Tables**:
  - `dim_users`: Stores user details (username, department).
  - `dim_event_types`: Stores event categories and descriptions.

Advantages:

- Simple design, easy to query.
- Optimized for reporting and dashboard visualization.
- Faster query performance due to fewer joins.

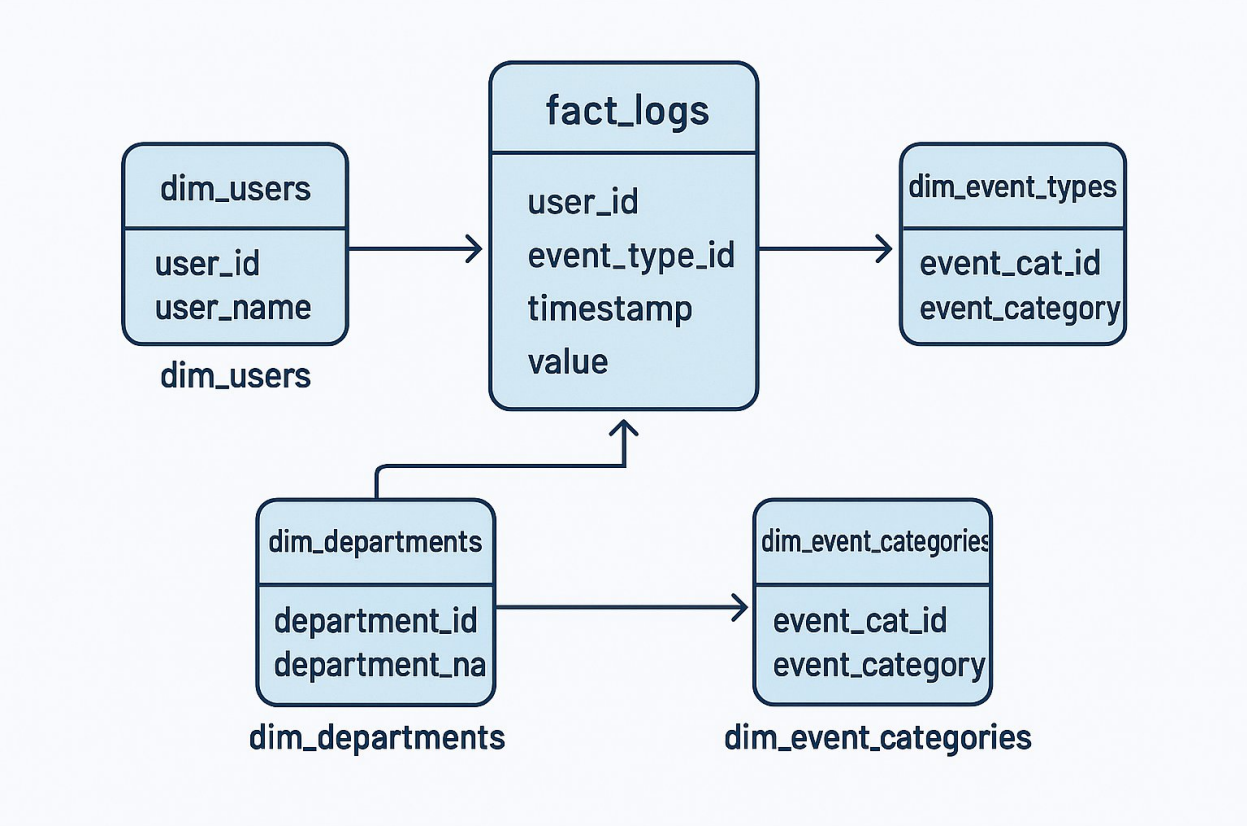


Snowflake Schema

- **Fact Table**: `fact_logs` (same as Star Schema).
- **Dimension Tables**:
  - `dim_users`: Linked to `dim_departments`.
  - `dim_event_types`: Linked to `dim_event_categories`.
  - `dim_departments`: Stores department details.
  - `dim_event_categories`: Stores event categories.

Advantages:

- Normalized structure reduces redundancy.
- Better suited for complex queries and large-scale data.
- Easier to maintain consistency across dimensions.



Schema Choice

- The **Star Schema** was used for **dashboard queries** and **reporting**, where speed and simplicity are critical.
- The **Snowflake Schema** was used for **detailed analysis** and **governance**, where normalization and consistency are more important.

SQL Queries & Optimization

Efficient querying of log data is essential for timely detection of anomalies and security threats. This project leverages **advanced SQL techniques** to enhance query performance and analytical capabilities.

Common Table Expressions (CTEs)

CTEs simplify complex queries by breaking them into modular components. **Example:**

sql

```
WITH high_severity AS (
    SELECT user_id, COUNT(*) AS event_count
    FROM fact_logs
    WHERE severity = 'HIGH'
    GROUP BY user_id
)
SELECT u.username, h.event_count
FROM high_severity h
JOIN dim_users u ON h.user_id = u.user_id;
```

Use Case: Identify users with frequent high-severity events.

Window Functions

Window functions allow trend analysis and anomaly detection across time. **Example:**

sql

```
SELECT user_id, event_time,
       COUNT(*) OVER (PARTITION BY user_id ORDER BY event_time ROWS BETWEEN 5
PRECEDING AND CURRENT ROW) AS recent_activity
FROM fact_logs;
```

Use Case: Track recent activity patterns per user.

Indexing

Indexes improve query speed by reducing scan times. **Example:**

sql

```
CREATE INDEX idx_event_time ON fact_logs(event_time);
```

Use Case: Faster retrieval of logs based on event time.

Partitioning

Partitioning divides large tables into manageable segments. **Example (Snowflake):**

sql

```
CREATE TABLE fact_logs_partitioned (
    log_id INT,
    user_id INT,
    event_time TIMESTAMP,
    event_type VARCHAR(50),
    severity VARCHAR(20),
    source_ip VARCHAR(50),
    destination_ip VARCHAR(50)
)
PARTITION BY (severity);
```

Use Case: Efficient queries on severity-based subsets of logs.

Performance Tuning Strategies

- Use **CTEs** for modular queries.
- Apply **indexes** on frequently queried columns.
- Implement **partitioning** for large datasets.
- Optimize joins by normalizing dimensions (Snowflake schema).
- Leverage **clustering** in Snowflake for faster query execution.

ETL/ELT Pipeline

Efficient data ingestion and transformation are critical for building a reliable cybersecurity log monitoring system. The project implements **Python-based ETL/ELT pipelines** to automate the process of collecting, cleaning, and loading log data into the Snowflake data warehouse.

Extract

The **extract phase** retrieves raw log data from multiple sources such as JSON files, CSV files, or streaming inputs. **Script:** `extract.py`

python

```
import json

def extract_logs(file_path):

    with open(file_path, 'r') as f:

        logs = [json.loads(line) for line in f]

    return logs
```

Function: Reads raw logs from a file and converts them into Python dictionaries.

Transform

The **transform phase** standardizes and enriches the raw data, ensuring consistency across fields. **Script:** `transform.py`

python

```
def transform_logs(logs):

    transformed = []

    for log in logs:

        transformed.append({

            "user_id": log.get("user_id"),

            "event_time": log.get("timestamp"),

            "event_type": log.get("event"),

            "severity": log.get("severity"),

            "source_ip": log.get("src_ip"),

            "destination_ip": log.get("dst_ip")
```

```
        })

    return transformed
```

Function: Cleans and maps raw log fields into a structured format suitable for loading into the warehouse.

Load

The **load phase** inserts transformed data into the Snowflake warehouse for storage and analysis.
Script: `load.py`

python

```
import snowflake.connector

def load_to_snowflake(transformed_logs):

    conn = snowflake.connector.connect(

        user='USERNAME',

        password='PASSWORD',

        account='ACCOUNT'

    )

    cursor = conn.cursor()

    for log in transformed_logs:

        cursor.execute("""

            INSERT INTO fact_logs (user_id, event_time, event_type, severity,
source_ip, destination_ip)

            VALUES (%s, %s, %s, %s, %s, %s)

        """, (log["user_id"], log["event_time"], log["event_type"],
log["severity"], log["source_ip"], log["destination_ip"]))

    conn.commit()

    cursor.close()

    conn.close()
```

Function: Loads structured logs into the `fact_logs` table in Snowflake.

Pipeline Workflow

1. **Extract** raw logs from files or streams.
2. **Transform** logs into a standardized schema.
3. **Load** logs into Snowflake for storage and analysis.

This pipeline ensures that data flows seamlessly from source systems into the warehouse, enabling both real-time and historical analysis.

Apache Spark Processing

Apache Spark is a powerful distributed processing framework that enables large-scale data analysis. In this project, Spark is used for both **batch processing** of historical logs and **streaming processing** of real-time log data.

Batch Processing

Batch processing is used to analyze historical log files stored in JSON or CSV format. **Script:** `batch_processing.py`

python

```
from pyspark.sql import SparkSession


spark = SparkSession.builder.appName("CyberLogBatch").getOrCreate()


logs = spark.read.json("logs.json")

logs.createOrReplaceTempView("logs")


result = spark.sql("""

    SELECT severity, COUNT(*) AS count

    FROM logs

    GROUP BY severity

""")


result.show()
```

Function: Aggregates logs by severity level to identify patterns and trends in historical data.

Streaming Processing

Streaming processing enables real-time monitoring of incoming logs. **Script:** `streaming_processing.py`

python

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("CyberLogStreaming").getOrCreate()

logs = spark.readStream.format("socket").option("host",
"localhost").option("port", 9999).load()

parsed_logs = logs.selectExpr("CAST(value AS STRING) as json")

query =
parsed_logs.writeStream.outputMode("append").format("console").start()

query.awaitTermination()
```

Function: Reads logs from a socket stream and outputs them continuously to the console for real-time monitoring.

Benefits of Spark Integration

- **Scalability:** Handles large volumes of log data efficiently.
- **Flexibility:** Supports both batch and streaming workloads.
- **Speed:** In-memory computation accelerates analysis.
- **Integration:**

Snowflake Features

Snowflake serves as the central data warehouse for storing and analyzing cybersecurity logs. Its advanced features make it well-suited for handling large-scale, complex, and semi-structured data while ensuring performance and security.

Clustering

Clustering organizes data within Snowflake tables to improve query performance.

- Logs are clustered by **severity** and **event_time**.
- Queries targeting specific severity levels or time ranges execute faster.
- Reduces scan costs by minimizing the amount of data read.

Example:

sql

```
CREATE TABLE fact_logs_clustered (
    log_id INT,
    user_id INT,
    event_time TIMESTAMP,
    event_type VARCHAR(50),
    severity VARCHAR(20),
    source_ip VARCHAR(50),
    destination_ip VARCHAR(50)
)
CLUSTER BY (severity, event_time);
```

Time Travel

Snowflake's **Time Travel** feature allows querying historical states of data.

- Useful for forensic analysis of logs.
- Enables recovery of accidentally deleted or modified records.
- Supports compliance by maintaining historical audit trails.

Example:

sql

```
SELECT * FROM fact_logs AT (TIMESTAMP => '2026-02-01 10:00:00');
```

Semi-Structured Data

Cybersecurity logs often contain JSON or other semi-structured formats. Snowflake natively supports these formats.

- JSON logs can be ingested directly.
- Queries can extract nested fields using Snowflake's `VARIANT` data type.

Example:

sql

```
CREATE TABLE raw_logs (log VARIANT);


SELECT log:user_id, log:event, log:severity

FROM raw_logs

WHERE log:severity = 'HIGH';
```

Benefits of Snowflake Integration

- **Scalability:** Handles large volumes of log data.
- **Flexibility:** Supports both structured and semi-structured formats.
- **Resilience:** Time Travel ensures data recovery and compliance.
- **Performance:** Clustering and partitioning optimize query execution.

Security Implementation

Security is a critical component of the Cybersecurity Log Monitoring System. Since the system deals with sensitive log data, it must ensure that only authorized users have access, sensitive information is protected, and compliance standards are met. The project implements multiple layers of security within Snowflake and the overall architecture.

Role-Based Access Control (RBAC)

RBAC ensures that users only have access to the data and functionality necessary for their role.

- **Admin Role**: Full privileges, including schema modifications and unrestricted queries.
- **Analyst Role**: Read-only access to log data for analysis.
- **Auditor Role**: Access to compliance-related queries and historical data.

Example:

sql

```
CREATE ROLE analyst;

CREATE ROLE admin;

GRANT SELECT ON fact_logs TO analyst;

GRANT ALL PRIVILEGES ON fact_logs TO admin;
```

Data Masking

Sensitive information such as IP addresses must be protected from unauthorized users. Snowflake's **dynamic data masking** ensures that sensitive fields are hidden unless the user has the appropriate role.

Example:

sql

```
CREATE MASKING POLICY mask_ip AS (val STRING) RETURNS STRING ->

    CASE

        WHEN CURRENT_ROLE() IN ('admin') THEN val

        ELSE 'XXX.XXX.XXX.XXX'

    END;
```

```
ALTER TABLE fact_logs ALTER COLUMN source_ip SET MASKING POLICY mask_ip;
```

Effect: Analysts see masked IPs, while admins see the original values.

Governance Policies

Governance ensures compliance with organizational and regulatory standards.

- **Audit Trails**: Snowflake's Time Travel feature maintains historical records.
- **Access Policies**: Define which roles can query specific tables.
- **Data Retention Policies**: Ensure logs are stored for compliance periods.

Benefits of Security Implementation

- Protects sensitive data from unauthorized access.
- Ensures compliance with cybersecurity and data privacy regulations.
- Provides transparency through audit trails.
- Supports role-based workflows for admins, analysts, and auditors.

Dashboard

The final layer of the Cybersecurity Log Monitoring System is the **interactive dashboard**, which provides real-time visibility into log data. Dashboards transform raw data into actionable insights, enabling analysts and administrators to quickly identify anomalies, trends, and potential threats.

Dashboard Features

- **Severity Distribution Charts**: Visualize the frequency of events by severity (Low, Medium, High).
- **Event Type Analysis**: Track which event categories occur most frequently.
- **User Activity Monitoring**: Identify users with unusual activity patterns.
- **Real-Time Streaming View**: Display incoming logs as they are processed by Spark.
- **Drill-Down Capabilities**: Allow analysts to explore specific events or users.

Implementation

The dashboard was implemented using **Plotly Dash** (Python-based), but can also be integrated with **Tableau** or **Power BI** for enterprise use.

Example Code (Plotly Dash):

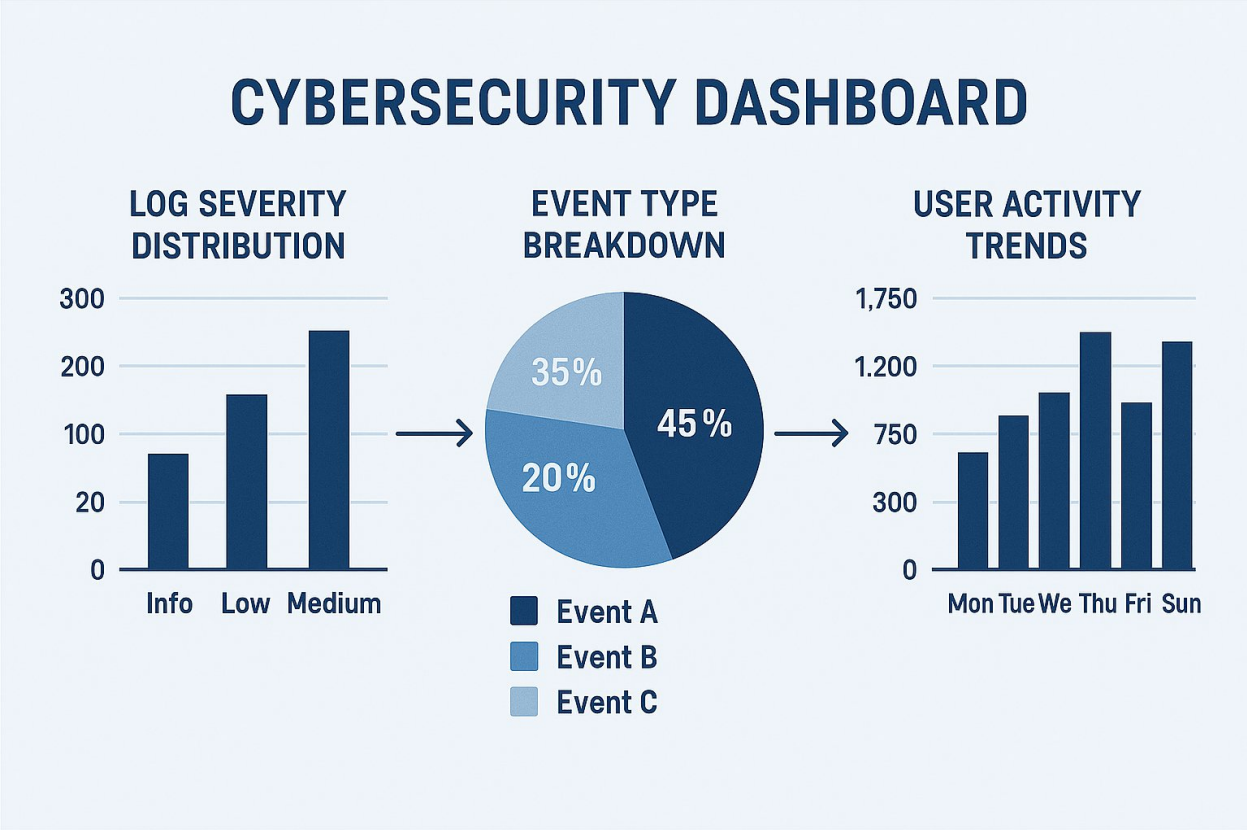
python

```
import pandas as pd

import plotly.express as px

df = pd.read_csv("logs.csv")

fig = px.histogram(df, x="severity", color="event_type", title="Log Severity
Distribution")

fig.show()
```

Function: Displays a histogram of log severity levels, grouped by event type.

Benefits

- **Real-Time Monitoring**: Provides immediate visibility into ongoing events.
- **User-Friendly Interface**: Simplifies analysis for non-technical users.
- **Decision Support**: Helps administrators prioritize responses to high-severity events.
- **Scalability**: Can be extended to include predictive analytics and alerts.


CYBERSECURITY DASHBOARD
LOG SEVERITY DISTRIBUTION
300
200
100
20
0
Info
Low
Medium
EVENT TYPE BREAKDOWN
35%
45%
20%
Event A
Event B
Event C
USER ACTIVITY TRENDS
1,750
1.200
750
300
0
Mon
Tue
We
Thu
Fri
Sun

Performance Tuning and Optimization

Performance optimization is essential for ensuring that the Cybersecurity Log Monitoring System can handle large volumes of log data efficiently while delivering timely insights. The project implements multiple strategies across SQL, Snowflake, and Apache Spark to achieve high performance.

SQL Optimization

- **Indexing**: Applied on frequently queried columns such as `event_time` and `severity` to reduce scan times.
- **Partitioning**: Divided large tables into severity-based partitions for faster queries.
- **CTEs**: Used to simplify complex queries and reduce redundant computations.
- **Window Functions**: Optimized for trend analysis without excessive joins.

Snowflake Optimization

- **Clustering**: Implemented clustering on `severity` and `event_time` to improve query performance.
- **Time Travel**: Used for efficient historical queries without duplicating data.
- **Semi-Structured Data Handling**: Leveraged Snowflake's `VARIANT` type to query JSON logs directly, avoiding costly transformations.
- **Caching & Micro-Partitioning**: Snowflake automatically optimizes storage into micro-partitions, reducing query latency.

Apache Spark Optimization

- **Batch Jobs**: Optimized by caching frequently accessed datasets in memory.
- **Streaming Jobs**: Configured with checkpointing to ensure fault tolerance and minimize recomputation.
- **Parallelism**: Increased the number of executors and partitions to balance workload across the cluster.
- **Predicate Pushdown**: Ensured filters are applied early in Spark jobs to reduce unnecessary data processing.

Overall Benefits

- Reduced query execution time by indexing and clustering.
- Improved scalability with partitioning and Spark parallelism.
- Enhanced resilience with checkpointing and Snowflake's time travel.
- Delivered faster insights for both real-time and historical log analysis.

Conclusion and Future Work

Conclusion

The Cybersecurity Log Monitoring System successfully demonstrates an end-to-end solution for monitoring, analyzing, and securing log data. By integrating a **data warehouse**, **advanced SQL queries**, **Python ETL pipelines**, **Apache Spark processing**, **Snowflake optimization**, and **security mechanisms**, the system provides both real-time and historical insights into cybersecurity events.

Key achievements include:

- Designing **Star and Snowflake schemas** for efficient data storage and querying.
- Implementing **advanced SQL queries** for anomaly detection and trend analysis.
- Building **Python ETL/ELT pipelines** to automate data ingestion and transformation.
- Leveraging **Apache Spark** for both batch and streaming log processing.
- Utilizing **Snowflake features** such as clustering, time travel, and semi-structured data handling.
- Enforcing **security policies** through RBAC, data masking, and governance.
- Developing an **interactive dashboard** for visualization and decision-making.
- Applying **performance tuning strategies** to optimize query execution and processing speed.

This system provides a scalable, secure, and efficient framework for cybersecurity log monitoring, supporting operational needs and compliance requirements.

Future Work

While the current implementation meets the project objectives, several enhancements can be explored to further strengthen the system:

1. **Machine Learning Integration**
    - Apply anomaly detection models to identify suspicious patterns.
    - Use predictive analytics to forecast potential threats.
1. **Automated Alerts & Notifications**
    - Integrate with messaging platforms (e.g., Slack, Teams, Email) to notify administrators of high-severity events in real time.
1. **Cloud-Native Deployment**
    - Deploy the system on cloud platforms (AWS, Azure, GCP) for scalability and resilience.
    - Use containerization (Docker, Kubernetes) for portability and orchestration.
1. **Enhanced Visualization**
    - Add drill-down dashboards with advanced filtering.
    - Integrate geolocation mapping for IP-based threat analysis.
1. **Compliance Extensions**
    - Implement modules for GDPR, HIPAA, and other regulatory frameworks.
    - Provide automated compliance reports.