

Building a Unified Security Event Pipeline for Enterprise Threat Detection

CS 4265: Big Data Analytics — Milestone 1: Project Foundation

Olaoluwa Adedamola Omodemi
College of Computing and Software Engineering
Kennesaw State University
Marietta, USA
oomodem2@students.kennesaw.edu

Abstract—As enterprise telemetry scales into the hundreds of millions of events, traditional security architectures face significant discrepancies between data capacity and processing throughput. This proposal outlines a distributed Big Data pipeline designed to ingest 700M+ records from the LANL Unified dataset. By integrating host logs, network flows, and external threat intelligence via a Medallion architecture, the system enables scalable detection of lateral movement. Key technologies include Apache Spark on YARN for parallel processing and HDFS/HBase for fault-tolerant storage and wide-column indexing.

Index Terms—Big Data, Apache Spark, Cybersecurity, HDFS, HBase, Threat Detection.

I. PROJECT OVERVIEW (DELIVERABLE 3.1.1)

A. Domain and Problem Statement

The domain is enterprise cybersecurity. Modern networks generate massive telemetry that exceeds the capacity of single-machine analysis. This project answers: *How can we correlate disparate authentication and network logs at scale to identify compromised credentials?* The primary scale challenge is the discrepancy between the volume of the **LANL dataset** (700M+ records) and the latency required for threat-hunting queries.

B. Scope

- **In-Scope:** Ingestion of LANL datasets; normalization via Spark DataFrames; correlation with threat feeds; and indexed storage in HBase.
- **Out-of-Scope:** Sub-second real-time alerting and proprietary enterprise network integration.

II. SYSTEM DESCRIPTION (DELIVERABLE 3.1.2)

A. Data Sources and 3-Vs Characteristics

- **Sources:** LANL User-Computer Auth logs (CSV), LANL Unified Host/Network logs (CSV), and URLHaus API threat feeds (JSON).
- **Volume:** 700M+ authentication events; 100M+ network flows.
- **Variety:** Heterogeneous formats including flat CSV files and semi-structured JSON API responses.
- **Velocity:** Batch-oriented ingestion with periodic scheduled updates for external threat intelligence.

B. Engagement with Stack Layers

This project engages every layer of the Big Data stack:

- **Storage Layer:** We utilize **HDFS** with a replication factor of 3 and 128MB **blocks** to ensure fault tolerance.
- **Syntax Layer:** Data is encoded into **Parquet** with Snappy compression for optimized columnar serialization.
- **Data Stores:** We implement a **Wide-column model** using **Apache HBase**, specifically managing **column families** to separate log data from enrichment metadata.
- **Processing Layer:** We use **Spark on YARN**. While Spark handles execution, we explicitly manage the **Shuffle** and **Reduce** phases during log correlation.
- **Querying Layer:** Analysts utilize **Spark SQL** for distributed execution, leveraging **Partitioning Strategies** by date to limit I/O.

C. Simplifying Assumptions

1) All datasets are de-identified. 2) Network latency within the Spark cluster is negligible compared to I/O. 3) Threat intel feeds are static during a single batch execution.

III. IMPLEMENTATION APPROACH (DELIVERABLE 3.1.3)

A. Technology Choices and Processing Model

We utilize a **Batch Architecture**. Technologies include PySpark for processing, HDFS for storage, HBase for the wide-column store, and Jupyter for the user interface.

B. Scalability Plan

To handle increasing data volume, the system utilizes horizontal scaling. Horizontal scaling is achieved by adding DataNodes to the HDFS cluster and additional Spark Executors. The **LSM Trees** in HBase ensure high-throughput writes as volume grows, while **Partition Pruning** ensures query performance remains constant over time.

C. Metrics for Success

We will measure: 1) **Throughput** (Events processed/sec), 2) **Latency** (Total batch time), and 3) **Data Volume** successfully ingested and enriched.

IV. LITERATURE REVIEW (DELIVERABLE 3.2)

- **MapReduce [1]:** Foundation for parallel execution. *Adaptation:* We adapt the shuffle phase to handle data skew in authentication logs.
- **HDFS Architecture [2]:** Enables distributed storage. *Adaptation:* Configuration of 128MB block sizes to optimize sequential log scans.
- **Bigtable/HBase [3]:** Wide-column design. *Adaptation:* Designing row-keys to manage **HBase regions** and prevent hotspotting during write-heavy security ingestion.
- **Spark DataFrames [4]:** In-memory abstraction. *Adaptation:* Using **Lazy Evaluation** to optimize the transformation DAG before 700M-record joins.
- **Bloom Filters [5]:** Space-efficient membership testing. *Adaptation:* We implement **Bloom Filters** to check if an IP is in the threat feed before triggering an expensive distributed join.

REFERENCES

- [1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing," OSDI, 2004.
- [2] K. Shvachko, "The Hadoop Distributed File System," IEEE MSST, 2010.
- [3] F. Chang et al., "Bigtable: Distributed Storage," ACM TOCS, 2008.
- [4] M. Zaharia et al., "Spark: Cluster Computing," HotCloud, 2010.
- [5] B. H. Bloom, "Space/Time Trade-offs in Hash Coding," Commun. ACM, 1970.



Fig. 1. Stack Architecture: Multi-layered Big Data stack from Distributed Storage to User Interface (3.3.1).

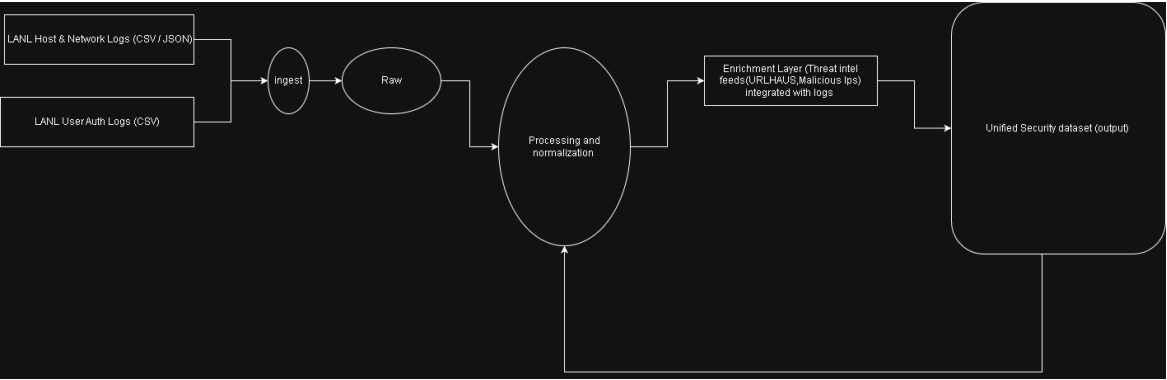


Fig. 2. Data Flow Pipeline: Specific processing logic showing $\times N$ parallel worker markers (3.3.4).