

Towards a Secure and Intelligent Access Control Policy Adapter for Big Data Environment

El Mostapha Chakir¹^a, Marouane Hachimi^{1,3} and Mohammed Erradi²

¹*HENCEFORTH, Rabat, Morocco*

²*ENSIAS, Mohammed V University, Rabat, Morocco*

³*INPT, Rabat, Morocco*

Keywords: Access Control, Policy Adaptation, Time Series, Big Data, Machine Learning.

Abstract: In today's digital landscape, Big Data is crucial for business efficiency and decision-making, but it raises significant Access Control challenges due to its growing scale, complexity, and diversity of user interactions. These challenges include ensuring data integrity, maintaining privacy, and preventing unauthorized access, all of which become increasingly difficult as data volumes and access points expand. In this paper, we propose an approach that combines Time Series Anomaly Detection with Machine Learning (ML) to enable adaptive Access Control policies that dynamically adjust based on detected anomalies and changing user behaviors in Big Data environments. By analyzing collected logs, we extract models of users' behaviors, which are then utilized to train an ML model specifically designed to identify abnormal behavioral patterns indicative of potential security breaches or unauthorized access attempts. The Access Control Policy Adapter uses the anomalies identified by the ML model, along with static and behavioral anomaly detection techniques, to adjust Access Control policies, thus ensuring that the system remains robust against evolving threats. We validate this approach using a synthetic dataset, and initial results demonstrate the effectiveness of this method, underscoring its potential to significantly enhance data security in complex Big Data ecosystems.


1 INTRODUCTION

In today's digital age, Big Data is crucial for businesses of all types. It improves operational efficiency and facilitates data-driven decision-making (John and Misra, 2017). However, the rapid increase in data volume makes it difficult to manage permissions effectively across increasingly large data, often resulting in either overly permissive access or restrictive controls that hinder legitimate data usage. Additionally, the speed of data generation demands real-time access decisions, which traditional access control systems such as discretionary access control (DAC) and role-based access control (RBAC) struggle to accommodate, potentially leading to bottlenecks or security vulnerabilities (Shan et al., 2024). As data increases in size and complexity, securing access to it becomes essential to maintaining the integrity and confidentiality of information systems.

Big Data environments are inherently dynamic and require equally dynamic access control systems

(Jiang et al., 2023). Throughout the life cycle of a big data resource, from its creation to its deletion, the access rights of different users must evolve. Consider the case in a financial services organization where an anomaly detection system identifies unusual data access patterns during non-business hours—a potential indicator of a data breach. Under traditional access control systems, adapting the access rights to temporarily restrict data visibility until the anomaly is investigated would require manual intervention, which is not feasible outside regular working hours (Karimi et al., 2021). This delay in response could lead to data leakage or other security breaches.

Within the Hadoop ecosystem, the leading big data management platform, Apache Ranger plays a central role in implementing robust access control through models such as attributes (ABAC) (Shan et al., 2024). However, Apache Ranger cannot dynamically adjust policies in response to constant changes, or identified anomalies, especially in the big data environment where access is frequently changed or the number of users is huge. There is an urgent need for access control mechanisms that can adapt

^a <https://orcid.org/0000-0001-7944-6344>

in real time to changes in data attributes and user roles, ensuring that security measures keep pace with rapidly changing data and user interactions (Walter, 2023).

To meet this requirement, recent research has increasingly focused on creating adaptive access control systems (Shan et al., 2024; Jiang et al., 2023; Karimi et al., 2021). These systems use machine learning and real-time analytics to automatically adjust access policies in response to changing data patterns, user behavior, and the emerging threat landscape. These adaptive approaches represent a major advancement in big data security, providing proactive strategies to identify and mitigate potential security breaches before they occur.

In this direction, this work introduces a novel approach within the Hadoop ecosystem, employing Machine Learning and Time Series Anomaly Detection to enhance Access Control security. By continuously monitoring data patterns, user interactions, and security threats, this method aims to improve Big Data environments' security dynamically.

This work focuses on dynamically updating Access Control policies based on real-time evaluations of user and system behavior. It achieves this by analyzing Apache Ranger audit logs, which are essential for detecting policy violations and analyzing user/system behavior. Rigorous testing has proven the effectiveness of this work in improving the security of Hadoop environments.

The main contributions of this research are summarized as follows:

- Implements a behavioral model using Apache Ranger logs to detect policy violations in real-time and analyze behavior.
- Develops a model that uses machine learning and time series anomaly detection to adapt access control policies based on anomalies detected in user/system behavior.
- The effectiveness of this work to improve big data security is verified through extensive testing and evaluation.

2 RELATED WORK

Traditional access control models cannot automatically adjust permissions when an object's state changes (e.g., a document being edited). Models like DAC and RBAC rely on static object names or identifiers, meaning access policies do not adapt even if the object's version or state changes (Basin et al., 2023). The Attribute-Based Access Control (ABAC)

model offers more dynamic permission management by adapting to changes in object attributes (Huang et al., 2022).

(Shan et al., 2024) proposed a method using heterogeneous graph neural networks to address redundancy in dependency paths and regional imbalance in provenance graphs for dynamic access control. This approach integrates community detection and key node identification within big data provenance graphs to efficiently generate lean provenance-based access control (PBAC) rules.

Another study by (Jiang et al., 2023) presented the SC-RBAC model that stands out by offering precise risk evaluation and adaptive access decisions. Demonstrated as effective through simulation tests, it acknowledges the need for future enhancements to address potential inaccuracies in access behavior due to misaligned goals, aiming to refine the control over doctors' access to medical data.

(Karimi et al., 2021) employed a reinforcement learning approach to dynamically adapt ABAC policies, leveraging user feedback and access logs. Results from testing on real and synthetic data suggest this method competes well with, and sometimes surpasses, conventional supervised learning approaches.

A heuristic solution to the NP-complete problem of adapting policies to ABAC using hierarchical attribute values was proposed in (Das et al., 2019). This solution uniquely incorporates environment attributes and highlights the limitation of needing matching attribute sets for policy migration, suggesting future exploration into ontology-based mapping and heuristic development for diverse attribute sets.

While existing research such as above, can significantly improve the field of access control, especially about ABAC models, machine learning applications, and policy adaptation mechanisms, they can't adjust dynamically the policies based on real-time analysis of user behavior and data patterns and not all tailored for Big Data environments (Premkamal et al., 2021).

3 ACCESS CONTROL IN BIG DATA ENVIRONMENTS

3.1 Access Control Challenges in Hadoop Big Data Ecosystem

The Hadoop ecosystem is a collection of open-source software projects that facilitate storing, processing, and managing big data. It provides a powerful and scalable platform for organizations to handle massive datasets that traditional data management tools strug-

gle with. Unfortunately, it presents many challenges for access control (Awaysheh et al., 2020). Its dynamic and distributed nature, with constantly arriving data and evolving user roles, renders traditional methods inadequate for granular control. Furthermore, basic Hadoop security features are insufficient. These limitations can lead to serious security risks, including unauthorized access, data breaches, reputational damage, legal issues, and even non-compliance with data privacy regulations (Shan et al., 2024; Gupta et al., 2017).

Solutions like Apache Ranger offer a robust ABAC solution for fine-grained access policies based on user attributes and data characteristics. As illustrated in Figure 1, Ranger plugins, integrated with Hadoop, enforce authorization, pulling user information from corporate directories to establish security policies.

Let $\mathcal{H} = \{\text{HDFS, Hive, HBase, Kafka, Knox}\}$ represent the set of all Hadoop components integrated with Apache Ranger plugins. Define U as the set of users, each with attributes A_u , and R as the set of resources, each with attributes A_r . The access control policies P are functions from user and resource attributes to access decisions, $\{\text{allow, deny}\}$. The authorization function \mathcal{A} , defined as $\mathcal{A}: U \times R \rightarrow \{\text{allow, deny}\}$, evaluates access permissions based on these policies:

$$\mathcal{A}(u, r) = \begin{cases} \text{allow} & \text{if } \exists p \in P : p(A_u, A_r) = \text{allow} \\ \text{deny} & \text{otherwise} \end{cases}$$

The enforcement of this function across the Hadoop components is encapsulated by:

$$\forall s \in \mathcal{H}, \forall u \in U, \forall r \in R : \mathcal{R}\mathcal{P}_s(u, r) = \mathcal{A}(u, r)$$

where $\mathcal{R}\mathcal{P}_s$ denotes the Ranger plugin associated with each Hadoop service s . This formulation compactly describes how Apache Ranger manages and enforces fine-grained access control within the Hadoop ecosystem.

Beyond its primary function of authorization, Ranger also comprehensively logs audit activities. The recorded audit data is invaluable for tracking and investigating specific actions within the system.

However, Ranger can be complex to set up and manage, requiring expertise in defining and maintaining access control policies. Additionally, its reliance on external services for authentication and authorization can introduce potential integration challenges which could lead to misconfigurations in policy management (Alzahrani et al., 2024). Such issues could potentially result in policy violations, security breaches, and other related vulnerabilities.

3.2 Problem Definition

Understanding the complexities of managing Apache Ranger, especially as users and data grow, is vital for strong Hadoop security (Gupta et al., 2017). Let's explore these challenges through a real-world example.

An organization uses Ranger for HDFS access control, restricting access to sensitive data. A misconfiguration in access control policies allows unauthorized access, emphasizing the importance of audit log analysis for identifying and fixing security gaps. Let:

- **Users (U):** The set of all users in the system. In this scenario, $U = \{\text{analyst, admin}\}$.
- **Resources (R):** The set of all resources that access control policies apply to. $R = \{\text{/user/data/financial, /user/data/marketing, /user/data/security, /user/data/management}\}$.
- **Access Types (A):** The types of access that can be granted to resources. $A = \{\text{READ, WRITE}\}$.
- **Policies (P):** The set of rules defining access permissions. Each policy $p_i \in P$ is a tuple (u, r, a) , indicating that user u has access type a to resource r ($u \in U, r \in R, a \in A$).

Let's suppose that the desired policy $p_d \in P$ grants read access only to the marketing.data folder for the data.analyst user:

$$p_d = \{(\text{analyst, /user/data/marketing, READ})\}$$

Let's Consider a scenario where the analyst has malicious intentions and has access to the HDFS system. The analyst could be attempting to gather information, expose data, or engage in other malicious intent.

Let L represent the audit log entries of Apache Ranger, which are essential for monitoring access and identifying potential security concerns. An audit log entry L is represented as a tuple (u, r, a, t, s) , where:

- u is the user who performed the access attempt,
- r is the resource that was accessed,
- a is the type of access attempted (e.g., READ or WRITE),
- t is the timestamp when the access attempt occurred,
- s is the success status of the access attempt (SUCCESS or FAILURE).

The access logs of the malicious user might look like

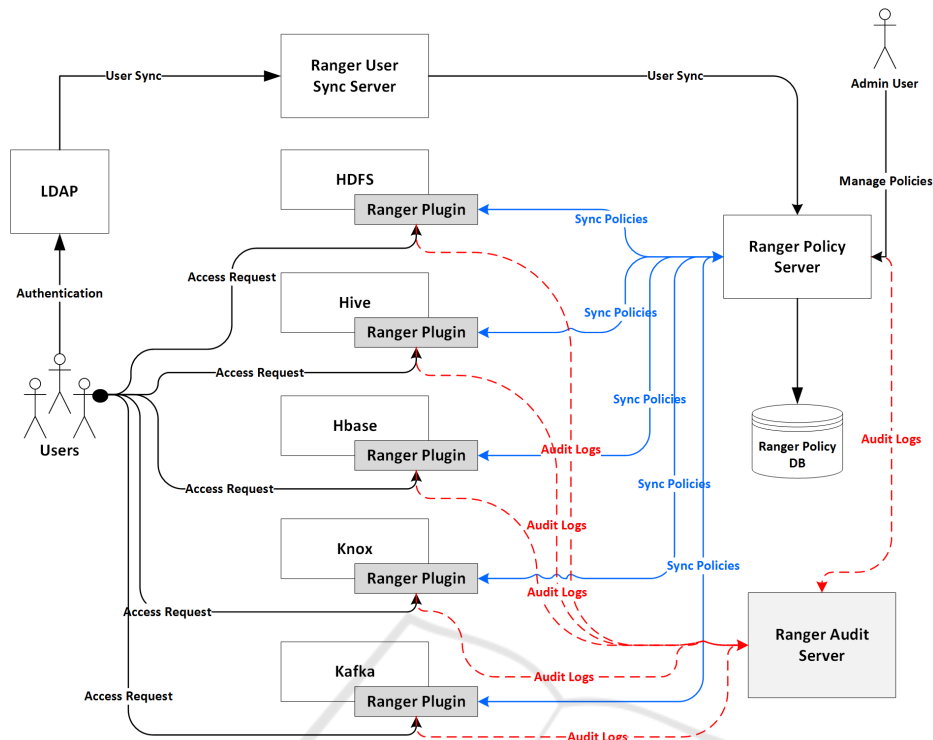


Figure 1: The working of the Apache Ranger.

the following:

$$L = \left\{ \begin{array}{l} (analyst, /user/data/marketing/report.csv, \\ READ, t_1, SUCCESS), \\ (analyst, /user/data/security/report.csv, \\ READ, t_2, FAILURE), \\ (analyst, /user/data/marketing/sales.csv, \\ READ, t_3, SUCCESS), \\ (analyst, /user/data/analyst/financial/report.csv, \\ READ, t_4, FAILURE), \\ (analyst, /user/data/management/report.csv, \\ READ, t_5, FAILURE) \end{array} \right.$$

Audit logs can reveal patterns of suspicious access attempts, but they rely on analyzing past user behavior. Security teams typically update policies manually after an incident. This highlights the need for continuous audit log monitoring and adaptive policies.

4 PROPOSED APPROACH

To address access control challenges in Hadoop, the proposed model leverages Apache Ranger’s audit logs. It uses behavioral monitoring analysis for real-time policy adjustments based on these audit logs. By proactively analyzing audit logs, the model strengthens Hadoop security, offering a sophisticated shield against vulnerabilities.

4.1 Architecture Overview

The proposed model introduces a straightforward approach for finding and fixing security issues in access control for big data environments. It uses the Apache Ranger Audit Log Server to track user activity, which helps in spotting problems. Figure 2 illustrates the different components and their interaction in the proposed approach.

We employ a multiple technique for anomaly detection. The *Initializer* sets up the system and ensures everything starts correctly. Central to the architecture is the *Ranger Audit Log Server*, which records all user activities. The *Retriever* continuously extracts logs from this server for user behavior monitoring. The *Behavioral Model* imports user behavior data and builds user-specific models to understand user conduct. *Detection Agents* analyze the system and identify anomalies. The *Cache Server* (Redis in this work) stores data retrieved by agents and provides it to the Policy Adapter. *Local Storage* stores data used for training the *ML Model*, which analyzes data to establish user-specific rate limits and detect suspicious behavior using machine learning. Finally, the *Policy Adapter* receives anomaly information from agents and updates Apache Ranger policies accordingly.

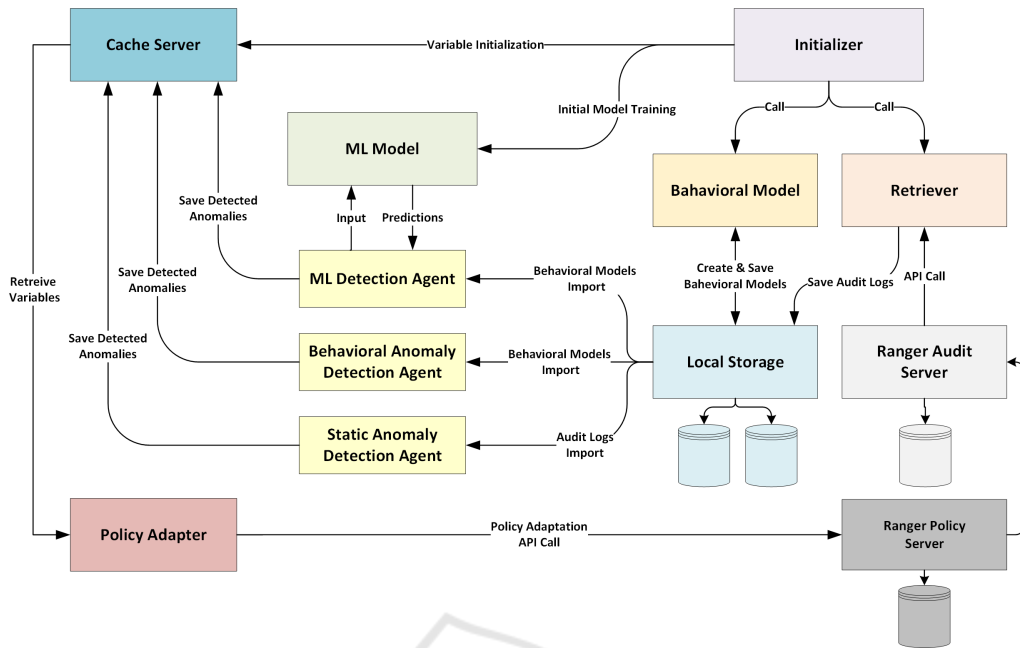


Figure 2: The suggested approach for Access Control Policy Adaptation.

We propose a robust algorithm for continuous monitoring and dynamic policy adaptation. It initializes essential variables and stores access logs and behavioral models in local storage. ML Model, particularly trained on this data, identifies and predicts anomalous patterns. The system updates periodically with new logs and refreshes models to reflect ongoing user interactions.

As new data arrives, the algorithm applies anomaly checks using both pre-defined rules and ML insights. These predictions should inform the policy adapter to dynamically adjust access control policies, mitigating risks identified by the anomalies.

Algorithm 1 illustrates the global interaction between all components.

Let's define the variables:

- \mathcal{C} : Cache variables initialized at the start of the system for storing intermediate data.
- \mathcal{L} : The complete set of access logs collected from the system for analysis.
- \mathcal{B} : The collection of all behavioral models built from the historical data representing user behavior patterns.
- \mathcal{A}_{static} : A set of detected static anomalies based on predefined rules or scores.
- $\mathcal{A}_{behavior}$: A set of detected behavioral anomalies based on deviations from the established behavioral models.
- $\mathcal{A}_{predicted}$: A set of predicted anomalies identified by the ML Model.

- \mathcal{P} : The current set of security policies, which can be adapted based on detected anomalies.
- \mathcal{M} : The machine learning model trained to detect anomalies.
- \mathcal{T} : A period or interval used to define how often the system should retrieve new logs and update models.

Input: First Time Setup

Output: Continuous Monitoring and Policy Adaptation

```

 $\mathcal{C} \leftarrow \text{Initialize}();$ 
 $\mathcal{L} \leftarrow \text{RetrieveAccessLogs}();$ 
 $\mathcal{B} \leftarrow \text{RetrieveBehavioralModels}();$ 
 $\mathcal{M} \leftarrow \text{TrainAIModel}(\mathcal{B});$ 
while true do
     $\mathcal{L}_{new} \leftarrow \text{RetrieveNewLogs}();$ 
     $\mathcal{B} \leftarrow \text{UpdateBehavioralModels}(\mathcal{L}_{new}, \mathcal{B});$ 
     $\mathcal{A}_{static} \leftarrow \text{StaticAnomalyCheck}(\mathcal{L});$ 
     $\mathcal{A}_{behavior} \leftarrow \text{BehavioralAnomalyCheck}(\mathcal{B});$ 
     $\mathcal{A}_{predicted} \leftarrow \text{PredictAnomalies}(\mathcal{A}_{static}, \mathcal{A}_{behavior});$ 
     $\mathcal{P} \leftarrow \text{AdaptPolicies}(\mathcal{A}_{predicted});$ 
end
    
```

Algorithm 1: Continuous Monitoring, Anomaly Detection and Policy Adaptation.

The used functions are defined as:

- $\text{Initialize}()$: Initializes the cache variables for the system.

- `RetrieveAccessLogs()`: Gathers access logs for analysis.
- `RetrieveBehavioralModels()`: Retrieves existing behavioral models from storage.
- `TrainAIModel(\mathcal{B})`: Trains the ML Model using the behavioral models as a dataset.
- `RetrieveNewLogs(\mathcal{T})`: Fetches new log entries that have been recorded since the last retrieval based on the period \mathcal{T} .
- `UpdateBehavioralModels(\mathcal{L}, \mathcal{B})`: Updates the behavioral models with new data from the access logs.
- `StaticAnomalyCheck(\mathcal{L})`: Identifies static anomalies in the new logs.
- `BehavioralAnomalyCheck(\mathcal{B})`: Detects behavioral anomalies by comparing new behaviors against established models.
- `PredictAnomalies($\mathcal{A}_{\text{static}}, \mathcal{A}_{\text{behavior}}$)`: Uses the trained ML Model to predict anomalies from the static and behavioral anomaly sets.
- `AdaptPolicies($\mathcal{A}_{\text{predicted}}$)`: Adjusts the security policies in response to the predicted anomalies.

4.2 Behavioral Model

The Behavioral Model builds user-specific models based on access logs from the Ranger Audit Log Server. It analyzes these logs to identify patterns and typical behaviors for each user. By capturing unique usage trends and access habits, the model creates a distinct profile for every user. This is achieved by focusing on relevant attributes like *id*, *serviceType*, *agentHost*, *clientIP*, *eventTime*, *eventDuration*, *accessResult*. Inspired by prior research (Argento et al., 2018), the Behavior Model is designed to generate individualized user profiles based on access logs. It systematically examines these logs to extract and organize behavioral data, identifying user-specific patterns and trends that reflect their interactions within the Big Data Environment. Algorithm 2 details the overall structure of the behavioral model.

Given a set of users U and their corresponding set of log entries L , we seek to construct a behavioral model B_u for each user $u \in U$. Each log entry $l \in L$ is a tuple:

$$l = (id, user, serviceType, agentHost, clientIP, eventTime, eventDuration, accessResult),$$

where each element represents a specific attribute of the log entry. The goal is to analyze and aggregate these log entries to model user behavior comprehensively.

Input: A set of access logs L
Output: A set of enhanced behavioral models $\{B_u^{\text{enhanced}}\}$ for each user $u \in U$

```

Initialize set of users:  $U \leftarrow \emptyset$ ;
foreach log entry  $l \in L$  do
    | Extract  $l_{\text{user}}$  and add it to  $U$ ;
end
foreach user  $u \in U$  do
    |  $L_u \leftarrow \{l \in L \mid l_{\text{user}} = u\}$  Extract and
    | aggregate features into
    |  $I_u, S_u, A_u, C_u, T_u, D_u, E_u, R_u$  Apply
    | transformation functions to generate  $B_u$ ;
end
foreach model  $B_u$  do
    |  $H_u \leftarrow \text{sort}(A_u), I_u \leftarrow \text{sort}(C_u)$  Enhance  $B_u$ 
    | by incorporating  $H_u$  and  $I_u$  into
    |  $B_u^{\text{enhanced}}$ ;
end
return  $\{B_u^{\text{enhanced}}\}$ 
    
```

Algorithm 1: Construction of User Behavioral Models.

Step 1: User-Specific Log Entry Aggregation: For each user $u \in U$, we identify the subset of logs L_u related to their activities by filtering operation:

$$L_u = \{l \in L \mid l_{\text{user}} = u\}.$$

Step 2: Feature Extraction: We extract features from each L_u to capture the user's behavioral patterns, defining sets for each attribute:

$$\begin{aligned}
 I_u &= \{l_{id} \mid l \in L_u\}, \\
 S_u &= \{l_{serviceType} \mid l \in L_u\}, \\
 A_u &= \{l_{agentHost} \mid l \in L_u\}, \\
 C_u &= \{l_{clientIP} \mid l \in L_u\}, \\
 T_u &= \{l_{eventTime} \mid l \in L_u\}, \\
 D_u &= \{l_{eventDuration} \mid l \in L_u\}, \\
 E_u &= \{l_{eventCount} \mid l \in L_u\}, \\
 R_u &= \{l_{accessResult} \mid l \in L_u\}.
 \end{aligned}$$

Step 3: Pattern Recognition and Model Formulation: The **Behavioral Model** B_u for each user u is an aggregation of the extracted features, formalized as:

$$B_u = \{I_u, \phi(S_u), \psi(A_u, C_u), \eta(T_u, D_u, E_u), \theta(R_u)\},$$

where ϕ , ψ , η , and θ are transformation functions that derive complex structures from the feature sets, such as frequency distributions, Cross-Reference of *agentHost* and *clientIP*, time series analyses, and statistical summaries, to provide insights into user behavior.

Step 4: Known Hosts and IPs Enhancement: The model is further enhanced by incorporating sorted lists of known hosts and IPs, adding contextual depth:

$$\begin{aligned} H_u &= \text{sort}(A_u), \\ I_u &= \text{sort}(C_u), \\ B_u^{\text{enhanced}} &= B_u \cup \{H_u, I_u\}. \end{aligned}$$

The aim is to capture the multifaceted aspects of user behavior from system interactions in a rigorous way.

4.3 Detection Agents

Detection agents perform a full range of system health checks, ensuring that any anomalies are quickly flagged for further investigation.

4.3.1 Static Anomaly Detection Agent

This agent is responsible for performing static anomaly checks on audit logs. It evaluates attributes such as *event duration* and *event count*, among others, to detect deviations from normal behavior that may indicate anomalies.

Let $L = \{l_1, l_2, \dots, l_n\}$ be the set of audit log entries, where each log entry l_i is defined as a tuple:

$$l_i = (id_i, aclEnforcer_i, eventCount_i, eventDuration_i)$$

where:

- id_i is the unique identifier for the log entry,
- $aclEnforcer_i$ specifies the ACL enforcement mechanism (e.g., 'ranger-acl', 'hadoop-acl'),
- $eventCount_i$ and $eventDuration_i$ are the key attributes scrutinized for anomalies.

Define A as the set of anomaly IDs, initially empty. For each log entry l_i in L , the following checks update the set A :

$$\begin{aligned} A &= \emptyset \\ \forall l_i \in L: & \begin{cases} A := A \cup \{id_i\} & \text{if } aclEnforcer_i \notin \{ 'ranger-acl', 'hadoop-acl' \} \\ A := A \cup \{id_i\} & \text{if } eventCount_i > 1 \\ A := A \cup \{id_i\} & \text{if } eventDuration_i > 0 \end{cases} \end{aligned}$$

After identifying anomalies, the agent synchronizes this data with the cache server.

The output of Static Anomaly Detection Agent's operation is the set A of anomaly IDs, which are synchronized with Redis, providing an updated and real-time reflection of system anomalies.

4.3.2 Behavioral Anomaly Detection Agent

This agent focuses on anomaly checks on generated behavioral models. It specifically looks for unusual logins from unknown IPs or hostnames. By monitoring login activities and comparing them to established user behavior patterns, It can detect any unauthorized access attempts or suspicious login patterns.

Given the set of all behavioral models B , where each model $b \in B$ corresponds to a user and is stored as a JSON file. Define A as the set of anomaly identifiers, initially empty.

For each behavioral model b stored in the directory 'Bh_Models', the following steps are taken:

1. Extract the user identifier usr from the model file-name.
2. Retrieve user information usr_inf from Redis database.
3. Load the behavioral data $data$ for usr .

$$\begin{aligned} usr &\leftarrow \text{extract}(b, '.json'), \\ \forall b \in \text{'Bh_Models'} : & \begin{aligned} &usr_inf \leftarrow \text{json.loads}(rds.get(usr)), \\ &data \leftarrow \text{load}(b), \end{aligned} \end{aligned}$$

$$\forall i \in \{0, \dots, \text{len}(data['serviceType']) - 1\} :$$

$$A := A \cup \begin{cases} \{data['ids'][i]\} & \text{if } data['serviceType'][i] \neq 'hdfs', \\ \{data['ids'][i]\} & \text{if } data['agentHost'][i] \notin usr_inf['known_hosts'], \\ \{data['ids'][i]\} & \text{if } data['clientIP'][i] \notin usr_inf['known_ips'] \end{cases}$$

After identifying anomalies, the Behavioral Anomaly Detection Agent synchronizes this data with the cache server.

4.3.3 Machine Learning Detection Agent

This agent is designed to perform anomaly detection on user behavior models using time series analysis of access logs (Ren et al., 2019). The main objective is to identify unusual patterns in denial events over time, which could indicate unauthorized access attempts or other forms of anomalous behavior. This process leverages machine learning techniques to analyze temporal variations in data and identify potential security threats.

Given a set of user behavior models generated by the behavioral model and stored as JSON files in local storage, let $B = \{b_1, b_2, \dots, b_n\}$ represent these models. Each model b_i contains sequences of log entries:

$$data_i = \{(t_1, r_1), (t_2, r_2), \dots, (t_m, r_m)\}$$

where t_j denotes the timestamp and r_j denotes the access result of each event.

For each user model, construct a time series T_i from the access denial events ($r_j = 0$):

$$T_i = \{(t_k, r_k) : r_k = 0\}$$

Calculate the hourly moving average of denials D_i^h for the time series T_i , which smooths the data over each hour h :

$$D_i^h(t) = \frac{1}{h} \sum_{k=t-h+1}^t r_k$$

Calculate the variations V_i in D_i^h to detect significant changes:

$$V_i(t) = D_i^h(t) - D_i^h(t-1)$$

Compute the mean μ and standard deviation σ of V_i :

$$\mu = \text{mean}(V_i), \quad \sigma = \text{std}(V_i)$$

Identify potential anomalies where the variation exceeds a threshold defined as three standard deviations above the mean:

$$\text{Anomalies} = \{t : V_i(t) > \mu + 3\sigma\}$$

After identifying anomalies, the ML Detection Agent synchronizes this data with a cache server (Redis).

4.4 ML Model

ML Model focuses on time-series anomaly detection using the Isolation Forest algorithm. This unsupervised learning technique excels at identifying outlying data points that deviate from typical patterns over time (Blázquez-García et al., 2021; Qin and Lou, 2019). It's well-suited for time-series data due to its random partitioning mechanism that naturally adapts to sequential data (Li and Jung, 2023).

Setting up and training the ML Model involves pre-processing steps specific to time-series data, such as normalization and extracting features like trends, seasonality, and autocorrelation. These steps prepare the data for the Isolation Forest algorithm (Xu et al., 2023).

Given a time-series dataset D where each data point x_t at time t is represented as a vector of features $\mathbf{x}_t \in \mathbb{R}^n$, the Isolation Forest algorithm seeks to identify points that are anomalies concerning the temporal distribution of the dataset.

4.4.1 Preprocessing

- **Feature Extraction:** Let $\mathbf{F}(\mathbf{x}_t)$ be a transformation that extracts relevant features from \mathbf{x}_t accounting for temporal properties such as lagged values, moving averages, and seasonality.

- **Normalization:** The features are normalized to ensure equal weighting during distance computations. If $\mathbf{F}'(\mathbf{x}_t)$ denotes the normalized feature vector, the normalization process can be represented as:

$$\mathbf{F}'(\mathbf{x}_t) = \frac{\mathbf{F}(\mathbf{x}_t) - \mu(\mathbf{F})}{\sigma(\mathbf{F})}$$

where $\mu(\mathbf{F})$ and $\sigma(\mathbf{F})$ are the mean and standard deviation of the features across the dataset.

4.4.2 Model Training

- Construct an ensemble of Isolation Trees, $\mathcal{T} = \{T_1, T_2, \dots, T_m\}$, from the transformed time-series dataset.
- For each tree T_i , a random subsequence of the time-series data is selected, and recursive partitioning is applied based on randomly selected features and split values.

4.4.3 Anomaly Score Calculation

The anomaly score for a data point \mathbf{x}_t is calculated based on the path length $h(\mathbf{x}_t)$ within each tree, averaged over the forest, and normalized as follows:

$$S(\mathbf{x}_t, n) = 2 \frac{E[h(\mathbf{x}_t)]}{c(n)}$$

where:

- $E[h(\mathbf{x}_t)]$ is the expected path length of \mathbf{x}_t over the forest \mathcal{T} .
- $c(n)$ is a normalization factor defined as the average path length in an unsupervised binary search tree given n external nodes.
- Shorter path lengths correspond to higher anomaly scores, indicating a higher likelihood of \mathbf{x}_t being an anomaly.

4.5 Policy Adapter

The Policy Adapter plays a critical role by dynamically adjusting Apache Ranger access controls in response to detected anomalies. It operates through key components: *Policy Retrieval*, which fetches existing policies for modification, and *IP Adaptation* and *Spike Adaptation* functions, which adjust policies to block unauthorized IPs and manage sudden access spikes, respectively. The adapter begins by retrieving anomalies from a Redis cache, linking these to specific policy IDs and user details from access logs. It then adapts policies based on the type of anomaly detected (Unknown IPs, Sudden Access Spikes etc.), using REST API calls to update these policies on the

Input: Anomalies from Redis \mathcal{A} , Access Logs \mathcal{L}
Output: Updated policies reflecting adapted security measures

```

Load environment variables;
Establish Redis connection;
Configure Ranger API credentials;
 $\mathcal{A} \leftarrow \text{Redis.hgetall('anomalies')}$ ;
 $\mathcal{L} \leftarrow \text{retriever.retrieve\_access\_logs}()$ ;
Initialize change_policies as empty dictionary;
foreach  $a \in \mathcal{A}$  do
  foreach  $\ell \in \mathcal{L}$  do
    if  $\ell.id = a.id$  then
      Prepare change request for  $a$ ;
      Add to change_policies;
    end
  end
end
foreach change  $c \in \text{change\_policies}$  do
  switch  $c.nature$  do
    case 'Unknown_IP' do
       $policy \leftarrow \text{retrieve\_policy}(c.policyId)$ ;
       $adaptation \leftarrow \text{IP\_policy\_adapt}(c)$ ;
      Update_policy(policy, adaptation);
    end
    case 'Deny_Spike' do
       $policy \leftarrow \text{retrieve\_policy}(c.policyId)$ ;
       $adaptation \leftarrow \text{Spike\_policy\_adapt}(c)$ ;
      Update_policy(policy, adaptation);
    end
    otherwise do
      // Handle other anomalies
    end
  end
end
return Updated policies

```

Algorithm 2: Dynamic Policy Adaptation Process.

Ranger server and handling responses to ensure updates are successful (See Algorithm 3).

Define the set of all policies as \mathcal{P} and the set of all detected anomalies as \mathcal{A} , where each anomaly $a \in \mathcal{A}$ is represented as a tuple $(id, nature, user, ip)$.

The policy retrieval function is defined as:

$$P : \mathbb{N} \rightarrow \mathcal{P}$$

This function $P(n)$ retrieves a policy using its identifier n , returning the policy as a structured object from

the Ranger server.

Define a function F that maps anomalies to policies:

$$F : \mathcal{A} \times \mathcal{P} \rightarrow \mathcal{P}$$

Function $F(a, p)$ applies transformations to policy p based on the anomaly a .

- IP-related anomalies adaptation:

$$\text{IPAdapt} : \mathcal{A} \rightarrow \mathcal{P}$$

Constructs modifications to policy p to handle unauthorized IP addresses based on the anomaly information.

- Access spikes adaptation:

$$\text{SpikeAdapt} : \mathcal{A} \rightarrow \mathcal{P}$$

Modifies p to temporarily deny user access in response to detected spikes.

For each anomaly a and corresponding policy p , execute the adaptation:

$$\forall a \in \mathcal{A}, p \in \mathcal{P} : \text{Execute}(F(a, P(\text{id}(a))))$$

The update function sends the adapted policy to the Ranger server and returns the status of the operation:

$$\text{Update} : \mathcal{P} \rightarrow \{\text{Success}, \text{Failure}\}$$

if $nature(a) = \text{'Unknown_IP'}$ then apply $\text{IPAdapt}(a)$
 if $nature(a) = \text{'Deny_Spike'}$ then apply $\text{SpikeAdapt}(a)$

5 IMPLEMENTATION

5.1 Dataset

To evaluate the proposed model, a synthetic dataset of 10,000 entries was created, simulating real-world Apache Ranger access logs. Each entry in the dataset represents an access event with attributes like service type, agent host, client IP, event time, duration, and result (permit or deny). This dataset ensures operational relevance with agent hosts and client IP addresses set to reflect typical settings. Each log entry is time-stamped during standard business hours, and distributed evenly across all days of the week, confirming a realistic workweek pattern. Randomization in the selection of the agent host and client IP address introduces variability, similar to the unpredictability of real logs. With a deny rate of approximately 14%, the dataset effectively emulates the decision-making process of an access control system.

Table 1 summarizes the attributes of the synthetic dataset. Each attribute is designed to mimic real-world access logs within a controlled environment.

Table 1: Attributes of the Synthetic Dataset used in the analysis.

Attribute	Description	Type
Service Type	Simulates HDFS interactions	Categorical
Agent Host	Indicates access node	Categorical
Client IP	Matches corresponding agent host	Categorical
Event Time	Time-ordered with added randomness	Temporal
Event Duration	Set to zero for simplicity	Numerical
Event Count	Represents a single transaction	Numerical
Access Result	Binary outcome (success or failure)	Categorical

5.2 Experiment

To evaluate the model’s anomaly detection, we developed the system using Python on a computer (i7-12800H CPU, 32GB RAM). We used default settings for the Isolation Forest algorithm to benchmark its performance.

Our methodology prioritizes data preparation for effective anomaly detection. Key features were extracted from the synthetic dataset, and a moving average calculation over a 3-hour window was applied to minimize noise and focus the model on significant anomalies. This step set the stage for accurate anomaly detection. PyCaret’s Anomaly Detection module was used to train the Isolation Forest model, enabling it to identify anomalies in new data based on an Anomaly Score.

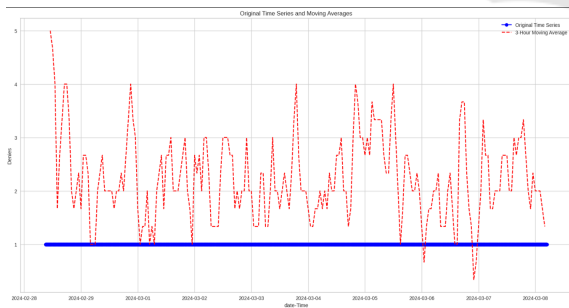


Figure 3: Time Series Analysis of Access Patterns with Moving Average Filtering.

We analyzed access denial patterns using time series analysis for the period February 28 to March 8, 2024 (Figure 3). The original data (blue line) represents individual access results (0 for success, 1 for denial), but this doesn’t reveal trends.

To address this limitation, we calculated a three-hour moving average (red dashed lines). This moving average represents the sum of access denials ev-

ery 3 hours, highlighting underlying trends in access attempts. This approach allows us to pinpoint periods with significant increases or decreases in access denial occurrences.

We applied the Isolation Forest algorithm to the access denial time series data. This algorithm excels at identifying anomalies, allowing us to pinpoint unusual activity (Figure 4). Green dots represent anomalies, where access density deviates significantly from the moving average trend. These anomalies could indicate potential security threats or system issues requiring investigation.

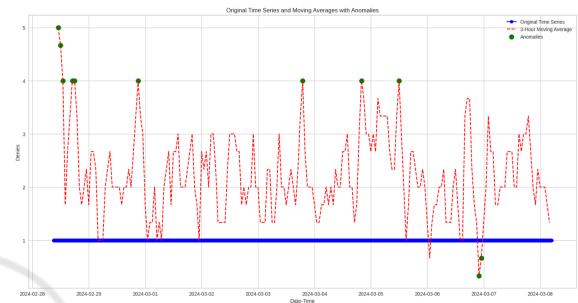


Figure 4: Anomaly Detection in Access Denial Events using Moving Average Analysis.

To improve the capabilities of the detection system, we use an additional experiment using a specialized anomaly detection agent with the same dataset that specifically targets behavioral access patterns. This agent uses machine learning techniques such as Isolation Forest, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN).

This approach is different from previous experiments, which primarily examined ”denied” access logs, instead focusing on temporal variations in access patterns to identify anomalies. Unlike previous methods, this new experiment introduces an agent focused on access time data rather than denial events. It undergoes extensive data preprocessing to normalize access patterns and establish a baseline standard. Then, the agent uses anomaly detection models with finely calibrated threshold parameters to improve detection accuracy while minimizing false positives.

$$\text{Anomaly Threshold} = \mu + 3\sigma$$

where μ is the mean of the anomaly scores, and σ is the standard deviation of the anomaly scores.

By analyzing the results of the three machine learning algorithms for anomaly detection in temporal access data, we observe the following performance characteristics:

The isolation forest model demonstrates a competent ability to identify anomalies with the adjusted anomaly threshold. In Figure 5, anomalies are repre-

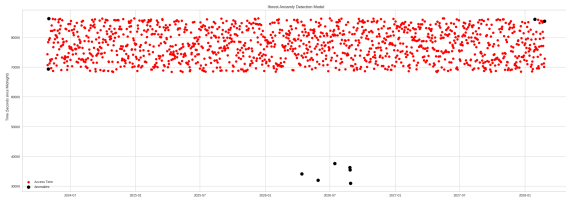


Figure 5: Anomaly Detection in Access Times Using Isolation Forest Algorithm.

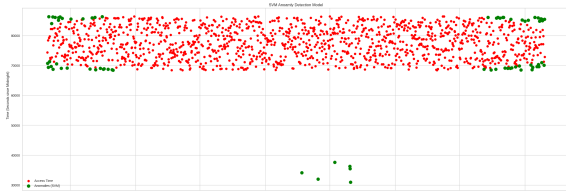


Figure 6: Anomaly Detection in Access Times Using SVM Algorithm.

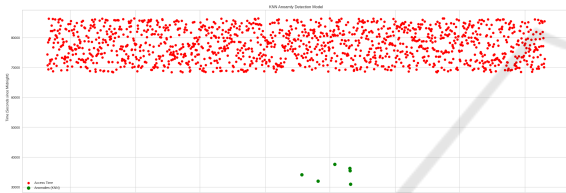


Figure 7: Anomaly Detection in Access Times Using KNN Algorithm.

sented as black dots, representing access times. Black dots are few in number and distinct from dense clusters of red dots, indicating a lower rate of false positives and accurate capture of true anomalies.

The SVM model, however, shows considerable green dots interspersed among the red dots throughout the timeline (Figure 6). This suggests that the SVM algorithm is reporting a significant number of false positives, as it is unable to effectively separate anomalies from normal data points despite adjustments to the anomaly threshold. The high frequency of green dots indicates poor discrimination between normal and abnormal data.

On the other hand, the KNN model outperforms the other two in terms of accuracy. Anomaly detection with KNN, marked by green dots, is sparse and very localized compared to the red dots (Figure 7). Adjusting the anomaly threshold as described before appears to have effectively minimized false positives, focusing only on the most statistically significant outliers.

Following anomaly detection, our system implements a process to dynamically adapt policies within Apache Ranger, effectively responding to various security threats as they arise. This process is streamlined through a meticulously designed algorithm as

described in section 4.5, that uses a two-phase approach: building a policy change dictionary and running a policy adaptation loop.

5.3 Discussion

To effectively manage policy adaptation in a Big Data environment, it is imperative to select an anomaly detection model that provides both high accuracy and efficiency. The analysis of the four models (Isolation Forest for temporal access logs and only access denied, as well as the implementations of K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) for the access logs temporal accesses provides a comprehensive overview of their ability to trigger specific policy adaptations.

Isolation forest demonstrates robust performance in identifying anomalies within a larger dataset of temporal access logs. Its strength lies in its ability to effectively separate anomalies from normal instances without being heavily influenced by noise present in the dataset, which is typical in large-scale data environments. Isolation forest applied specifically to access denials can be particularly effective in environments where unauthorized access attempts are a significant security issue, because it can detect subtle patterns of anomalous denials that broader models might overlook. KNN demonstrated exceptional accuracy in experiments, especially when finely tuned with an appropriate threshold. Its main limitation is the computational cost, which can increase with the size of the data. SVM, although comprehensive, has struggled with reliability and high false positive rates in experiments.

Our study on time series anomaly detection using machine learning suggests potential security improvements through Apache Ranger log analysis. This model identifies anomalies indicating possible access control policy violations, prompting necessary policy adaptations and suggesting areas for security hardening.

In access control, changing access requests over time makes it difficult for the model to rely only on the initial training data. Therefore, it is essential to continually update the model with new arriving logs and behaviors. To address this issue, we have chosen to focus on online learning in our future work, to improve this approach and ensure that the model remains effective and relevant in real-time scenarios.

6 CONCLUSION

We presented an approach that offers a comprehensive framework for anomaly detection in access control logs using time series analysis and machine learning. It combines static rules with behavioral patterns to identify unusual activity. Based on identified anomalies, the proposed system adapt automatically the Apache Ranger policies, core functionalities like caching, log storage, ML Model, and initial anomaly detection using different agents are operational and show promise.

To ensure our anomaly detection system stays adaptive and responsive, we plan to implement on-line learning techniques. This approach will allow our models to continuously learn and adjust from new data without the need for retraining, thereby maintaining their accuracy and effectiveness over time. This strategic focus not only aims to enhance security measures but also to adapt dynamically to ever-changing data landscapes, ultimately supporting robust and resilient access control policies.

REFERENCES

- Alzahrani, B., Cherif, A., Alshehri, S., and Imine, A. (2024). Securing big graph databases: an overview of existing access control techniques. *International Journal of Intelligent Information and Database Systems*.
- Argento, L., Margheri, A., Paci, F., Sassone, V., and Zannone, N. (2018). Towards adaptive access control. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 99–109. Springer.
- Alwaysheh, F. M., Alazab, M., Gupta, M., Pena, T. F., and Cabaleiro, J. C. (2020). Next-generation big data federation access control: A reference model. *Future Generation Computer Systems*, 108:726–741.
- Basin, D., Guarnizo, J., Krstic, S., Nguyen, H., and Ochoa, M. (2023). Is modeling access control worth it? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2830–2844.
- Blázquez-García, A., Conde, A., Mori, U., and Lozano, J. A. (2021). A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)*, 54(3):1–33.
- Das, S., Sural, S., Vaidya, J., and Atluri, V. (2019). Policy adaptation in hierarchical attribute-based access control systems. *ACM Transactions on Internet Technology (TOIT)*, 19(3):1–24.
- Gupta, M., Patwa, F., and Sandhu, R. (2017). Object-tagged rbac model for the hadoop ecosystem. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 63–81. Springer.
- Huang, H., Zhang, J., Hu, J., Fu, Y., and Qin, C. (2022). Research on distributed dynamic trusted access control based on security subsystem. *IEEE Transactions on Information Forensics and Security*, 17:3306–3320.
- Jiang, R., Han, S., Yu, Y., and Ding, W. (2023). An access control model for medical big data based on clustering and risk. *Information Sciences*, 621:691–707.
- John, T. and Misra, P. (2017). *Data lake for enterprises*. Packt Publishing Ltd.
- Karimi, L., Abdelhakim, M., and Joshi, J. (2021). Adaptive abac policy learning: A reinforcement learning approach. *arXiv preprint arXiv:2105.08587*.
- Li, G. and Jung, J. J. (2023). Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges. *Information Fusion*, 91:93–102.
- Premkamal, P. K., Pasupuleti, S. K., Singh, A. K., and Alphonse, P. (2021). Enhanced attribute based access control with secure deduplication for big data storage in cloud. *Peer-to-Peer Networking and Applications*, 14:102–120.
- Qin, Y. and Lou, Y. (2019). Hydrological time series anomaly pattern detection based on isolation forest. In *2019 IEEE 3rd information technology, networking, electronic and automation control conference (IT-NEC)*, pages 1706–1710. IEEE.
- Ren, H., Xu, B., Wang, Y., Yi, C., Huang, C., Kou, X., Xing, T., Yang, M., Tong, J., and Zhang, Q. (2019). Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3009–3017.
- Shan, D., Du, X., Wang, W., Wang, N., and Liu, A. (2024). Kpi-hgnn: Key provenance identification based on a heterogeneous graph neural network for big data access control. *Information Sciences*, 659:120059.
- Walter, M. (2023). *Context-based Access Control and Attack Modelling and Analysis*. PhD thesis, Dissertation, Karlsruhe, Karlsruher Institut für Technologie (KIT), 2023.
- Xu, H., Pang, G., Wang, Y., and Wang, Y. (2023). Deep isolation forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*.