Multiple-Criteria Query Statement Probabilities Based Database Insider Attack Monitoring System

CHEOLMIN SKY MOON, University of Washington Tacoma

ADVISERS:  
SAM CHUNG, University of Washington Tacoma  
BARBARA ENDICOTT-POPOVSKY, University of Washington Seattle

The purpose of this research is to propose a novel approach to monitor insider attack in dynamic database systems by query and query transition probabilities with multiple query statement preprocessing algorithms. Any malicious attack on the database systems performed by an entrusted group of people having authorized access is called database insider attack. Reliable research has shown that insider attacks are actually as dangerous as outsider attacks. Even though database insider attacks have been actively researched, most of the proposed approaches still have three common limitations keeping them far from the practical application: Firstly, they build a profile of the insider’s queries or transaction signatures to detect an anomalous query or transaction, but this requires profiling an insider’s behavioral factors to detect insider attack. Secondly, no query or transaction preprocessing algorithm can represent the query or transaction in a holistic perspective. Thirdly, their approach is not suitable for a dynamic database, where data updates are frequently happening. Thinking out-of-the-box, we propose a new approach which not only overcomes the previous three limitations, but also has architectural strength to deal with huge amounts of data monitoring. This approach has four objectives: (1) Multi-Preprocessing Algorithms to observe a query in multiple perspectives (2) a Query Probabilities Based Database Insider Monitoring Methodology based on Markov Mathematical Model to record insider’s behavioral patterns with query and query transition probabilities (3) a Query Probabilities Time Series Graph to monitor the insider’s behavior to predict insider attack (4) a Multi-Criteria Query Probabilities based Insider Attack Monitoring System that contains (1) – (3). The results from the evaluation show that the proposed system overcomes the described limitations and is capable to monitor an insider’s behavioral data in real time.

- Categories and Subject Descriptors: K.6.5 [Management of Computing and Information Systems]: Security and Protection TBD

General Terms: insider attack, database, query statement, SQL statement, transaction, monitoring, log, transaction probability

Additional Key Words and Phrases: distributed system, cloud computing, Cassandra DB, Markov Chain, statistics

ACM Reference Format:

TBD

1. INTRODUCTION

The definition of an insider attack, provided by the Computer Emergency Response Team (CERT) program at Carnegie Mellon University, is ‘a threat performed by a malicious insider who is a current or previous employee, or business partner having authorized access to the system or data, but intentionally uses that authority to harm the system or information [Silowash et al. 2012].’ According to the key findings from the 2013 State of Cybercrime Survey, 34% of the respondents answered that electronic crimes were more costly or destructive to the organization when caused by an insider compared to outsider, by as much as 31% [PwC 2013]. According to another survey conducted by the CERT Insider Threat Center, 43% of respondents...
experienced at least one malicious insider attack in 2010, and 46% of the respondents think the damage caused by insider attacks was more dangerous than the damage from outsider attacks [Silowash et al. 2012]. The surveys above indicate that insider attacks need to be treated as being at least as important as external threats.

There are several papers suggesting comprehensive insider attack detection models in abstracts without accompanying implementation details [Doss and Tejay 2009; Greitzer and Hohimer. 2011; Nithiyanandam et al. 2012]. The papers are not very helpful to implement an insider attack detection system in specifics, but are useful to comprehend the required components of an insider attack detection system and to understand how the components might work together. One of the critical components of the system mentioned in the three papers is a monitoring component: notably Greitzer mentioned many different kinds of log files that are meaningful to perform insider threats analysis [Greitzer and Hohimer. 2011].

Most of the papers dealing with a database insider attack detection system actually have a database query log monitoring component [Kamra et al. 2008; Chagarlamudi et al. 2009; Liu and Huang. 2009; Mathew et al. 2010; Raissi-Dehkordi and Carr. 2011; Rathod et al. 2012]. Chagarlamudi pointed out that the potentiality of insider attacks to a database is greater than outsider attacks; the more the legitimate insider knows about database systems, the greater the possibility of threats [Chagarlamudi et al. 2009]. Liu said that employees with legitimate access — such as database administrators, system administrators, application developers, and human resource management in the enterprise who have access to sensitive databases — there is more possibility of intentional or unintentional data corruption. For this reason, employees authorized to access critical information in a database should be closely monitored [Liu and Huang. 2009].

Although the database insider attack seems actively researched as indicated above, the systems proposed by previous research have three limitations. Firstly, all the previous work we have discovered focuses on profiling the signature of user’s queries or transactions. However, as Greitzer mentioned, we need to focus on an insider’s behavioral factors to support insider threat analysis [Greitzer and Hohimer. 2011]. Secondly, a data preprocessing algorithm takes on a critical role in detecting query or transaction pattern anomalies, but, to the best of our knowledge, one data preprocessing algorithm cannot perfectly identify a query statement in a holistic perspective. Accordingly, no preprocessing algorithm is able to perfectly detect query or transaction differences. For example, the two queries which are interpreted as the same by a preprocess algorithm could do different things [Mathew et al. 2010]. Lastly, the previous approaches are not suitable for a dynamic database whose internal state is dynamically changing. Mathew mentioned that data insertion is not allowed in a data-centric approach [Mathew et al. 2010]. Syntax-centric approaches did not mention this issue, but we assume that a valid set of data and user profiles should be updated every time a new valid access happens because if not, it will generate a high ratio of false positives. The time complexity of profiling calculation would not be trivial, while the size of data is getting bigger.

The purpose of this research is to suggest a design for a Multiple-Criteria Query Probabilities Based Database Insider Monitoring System. For this purpose, we asked the following three research questions:

- What data preprocessing methodology is available to, not only detect an anomalous query or transaction, but also monitor the insider’s behavioral
pattern which can be used for further analysis if we collect enough data using the methodology?

- What approach is available to observe a query or transaction in multiple perspectives?
- What approach is suitable to detect an anomalous query or transaction for the dynamic database? Whenever the database is changed, the current approaches need to keep up a new training phase and the time spent in the training phase would not be trivial.

Based on the considerations of ease and taking a structured approach to collect real world data that enables adequate scientific validation of proposed solutions for insider attack detection, we answer what data we need to collect and how to detect insider attacks. The implementation of our approach, a Multiple-Criteria Query Probabilities Based Database Insider Attack Monitoring System logically consists of five parts:

- Query probabilities based database insider monitoring system architecture: Since the proposed system would need relatively higher computing power and larger storage to store the probabilities based query log, we also propose an architecture for the monitoring system which provides the below features.
  - Capable to process large amounts of log data to provide real-time monitoring results,
  - Cloud-based system to distribute the workload of log processing and monitoring over multiple nodes,
  - Easy to scale.
- Multi-query statement preprocessing algorithms:
  - Operation Related Column Name (ORCN) based query statement preprocessing algorithm,
  - Query Command and Table Name (QCTN) based query statement preprocessing algorithm.
- Query probabilities based database insider monitoring methodology based on Markov mathematical model:
  - Query transition probability,
  - Query probability,
  - Probabilities based query log.
- Query probabilities time series graph,
- Multi-criteria query probabilities based insider attack monitoring system.

2. BACKGROUND

2.1 Insider Attack

Insider attack is a threat performed by a malicious insider who is a current or previous employee, or business partner having authorized access to the system or data, but intentionally uses that authority to harm the system or information [Silowash et al. 2012]. Insider threat also has the same meaning as insider attack [Greitzer and Hohimer. 2011]. Insider attacks are not recognized to the degree of external attacks, but they can be as dangerous as external attacks. The nature of problem has been shown for many years. The Department of Defense (DoD) reported that 87% of recognized intrusions into information systems were from employees or internal organizations in 1997 [Greitzer and Hohimer. 2011].

Due to the nature of insider threats, there are inherently more entry points for malicious attacks than those of outsider threats, especially with the possibility of physical access to the system [Greitzer and Hohimer. 2011]. For that reason, insider
threats should be separately considered from the beginning of threat detection system design.

2.2 General Query Log in MySQL
The existing security features for databases such as privileges, views, roles, etc. are not enough to protect the database from insider (both malicious and non-malicious) misuses [Chagarlamudi et al. 2009]. For that reason, the only feasible solution to monitor and detect the insider threat is to use database auditing [Liu and Huang, 2009]. Most of the major Database Management System (DBMS) provide a built-in auditing feature, and the log generated by this feature contains information of who accessed the database and what queries the user performed with a time stamp. MySQL has several built-in logging features: error log, general query log, binary log and slow query log. The general query log is used to log established client connections and statements received from clients [MySQL. 2014]. This feature seems very useful, but there is a severe performance drawback with the auditing feature. Giuseppe Maxia performed an experiment on MySQL 5.4 to test how much the auditing feature in MySQL harms database performance [Maxia. 2009]. The test shows that performance drops up to 54.4% for the database table log and 19.17% for the file log. Using a file log will increase performance by 77.3% over a table log for the database, but it would be more cumbersome to implement a parser for the log file and analyze it. Bypass of the auditing function does not affect performance of the database [Liu and Huang, 2009], but it can only be applied for accessing databases using the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP protocols are not applicable to the insider having physical accessibility to the database.

2.3 Syntax-Centric and Data-Centric Approaches
The syntax-centric approach has been discussed in many papers, but the term, syntax-centric, was first mentioned by Sunu Mathew, who proposed the data-centric approach [Mathew et al. 2010]. The two approaches are used to construct a user profile. The syntax-centric approach, which is commonly used, builds a user profile based on query statement syntax; however, the data-centric approach uses data attained from the result of a query [Mathew et al. 2010]. The data-centric approach turned out to be more accurate, but it has a critical drawback. Data is not allowed into the database being monitored. In other words, the data-centric approach can only be used for a static database, having a constant state.

2.4 Log Analysis in the Cloud
Database log data needs to be processed and stored in one place to be analyzed. We need to have log data for every user for several years for long term monitoring [DOSS, G. AND TEJAY, G. 2009], but the log data from databases in an enterprise cannot be processed in traditional distributed log data processing systems because the amount of data grows exponentially [Yu and Wang. 2012]. Fortunately, the scalability of cloud computing makes this mass log data process possible because we can easily scale up the log processing system in the cloud as log data grows [Wei et al. 2011].

3. PREVIOUS WORK
There were many papers proposing an insider attack detection system, but few papers have focused on database insider attack detection. For this reason, we researched not only database insider attack detection systems specifically, but also
general insider attack detection systems and database auditing systems concerning internal threats.

3.1 Detecting Anomalous Access Patterns in Relational Databases [Kamra et al. 2008]
Kamra proposed an anomaly detection method derived from the query logs of database access. They first preprocessed each query statement with three data representations: c-quiplet, m-quiplet and f-quiplet. Then they applied a clustering algorithm to construct profiles representing normal user behavior to detect anomalies. This research paper is for anomalous access detection, not insider attack detection, but it provided a starting point of our thinking about the insider attack detection problem.

3.2 An Insider Threat Prevention System on Database [Chagarlamudi et al. 2009]
The mechanism to detect threats in this system is based on a predefined task set for each user. The assumption of this system is that there are multiple applications performing tasks which consist of several transactions. Multiple transactions that were partially ordered with Petri Nets compose each task. The transactions are predefined by the applications, so no user can add, remove or modify the transactions. Whenever a user performs a task, the system checks whether the task is in the set allocated to the user, and the order of transactions to perform the task follows the pre-defined Petri net for each task; however, the application of this approach is very limited because it can only detect the insider threats performed by applications. Moreover, the schema of the DB is not allowed to update after the learning phase.

3.3 A Data-Centric Approach to Insider Attack Detection in Database Systems [Mathew et al. 2010]
This system uses a distinct approach from the previous four to identify a transaction. It uses a specific term, data-centric approach as opposed to syntax-centric (3.1, 3.2, 3.3 and 3.4). The authors argue that query syntax alone is not enough to know the insider’s intent; however, the observation of what data is accessed by the insider gives much better insight into the insider’s intent. Accordingly, it uses Statistic Vector features of the query result data to discover the pattern. The results showed significant improvement in most of the tests conducted. In spite of the improvement, the drawback of the data-centric approach is that data insertion is not allowed, so it can be used in only static databases, which is rare in the real world.

3.4 A Framework for Database Auditing Which Does Not Affect the Performance of the Database [Liu and Huang. 2009]
One of the main problems of database auditing using a DBMS auditing tool is that it drops the performance of the database up to 54.40% [Maxia. 2009]. It captures network packet and parses it to extract the Structured Query Language (SQL) transaction commands from each packet. In other words, auditing is done outside of the database, so it does not affect the performance of the database; however, it cannot be used to detect a malicious insider who has the ability to directly access the physical machine hosting database.

3.5 A Multi-Perspective Auditing Approach [Raissi-Dehkordi and Carr. 2011]
Database auditing is used in one of three dimensions: user, file and database metrics. A different supervised machine learning model, Support Vector Machine (SVM) is employed for each dimension to train the machines. User behavior analysis by the three SVMs supports the aggregate detect module to analyze the three views together. This paper also considers a malicious event performed by the aggregation of
the multiple normal events performed by the group of malicious users. It is useful to analyze user behaviors in multiple perspectives using the three dimensional components, but the paper lacks a detailed description of how to classify the attributes in each dimension to generate profile metrics.

3.6 Intrusion Detection in Database Based on Transaction Signature [Rathod et al. 2012]

The paper suggested an architecture that consists of three phases: the first phase is a learning phase - teaching the machine with offline log data which is considered as legitimate log data. During the second phase, a user’s signature is generated by a transaction performed by the user. The third phase is the response phase - the trained machine compares the user signature with legitimate transaction sets. As mentioned in the introduction, the signature based preprocessing algorithm could not detect every transaction difference and similarity. Also, the database schema is not permitted to update after the learning phase.

4. MOTIVATION

Synthesizing the previous work described above, we summarize the insider attack detection process in five steps [Kamra et al. 2008; Chagarlamudi et al. 2009; Mathew et al. 2010; Liu and Huang. 2009; Raissi-Dehkordi and Carr. 2011; Rathod et al. 2012].

(1) Decide a unit of data being monitored: First, the system should define a granular unit of data being monitored. Three unit types are used in the previous papers. Kamra used a query statement as the unit [Kamra et al. 2008]. Chagarlamudi and Rathod used a transaction [Chagarlamudi et al. 2009; Rathod et al. 2012]. Mathew used a result of a query execution as the unit [Mathew et al. 2010].

(2) Preprocess the unit of data: Each unit (1) is redefined (preprocessed) using focused attributes to identify the unit for constructing a user’s profile. Kamra identified a query using three data representation: c-quiplet, m-quiplet and f-quiplet [Kamra et al. 2008]. Chagarlamudi used partial order of transactions with Petri Nets to identify a set of transactions [Chagarlamudi et al. 2009]. Rathod redefined a transaction using the signature of a legitimate transaction: user name, transaction ID, total command, command sequence and time. Mathew identifies a query with the seven statistical vectors to reinterpret the result of a query.

(3) Train a set of preprocessed data: Construct a user’s profile with the set of preprocessed units of log data belonging to the user. In this process, we need to assume the data feed contains only normal behaviors of the user.

(4) Monitor the user’s activities: After the user’s profile is constructed, monitor the user’s queries or transactions in the database.

(5) Detect anomalies: If a query or transaction is not matched to the constructed user’s profile, the system recognizes it as an insider attack.

After considering details in each step proposed in previous work, we realized that the terms “insider attack” and “anomalous behavior detection” are interchangeably used in many papers. From that observation, we realized the general approach used for detecting anomalous access patterns suggested by Kamra is also used for detecting insider attack in all papers we surveyed [Chagarlamudi et al. 2009; Liu and Huang. 2009; Mathew et al. 2010; Raissi-Dehkordi and Carr. 2011; Rathod et al. 2012], however, the same approach used in anomalous access pattern detection cannot be used in insider attack detection because the anomalies they detect are
different from each other. The former detects a query or transaction anomaly, but the latter detects an insider’s behavioral anomaly. For that reason, the former would be a subset of the latter. To date, no research has attempted to address such human behavioral factors to support insider threat analysis [Greitzer and Hohimer, 2011]. The biggest reason is lack of sufficient real world data to understand normal versus anomalous behavior (not anomalous query or transaction) [Greitzer and Hohimer, 2011]. Before expanding the discussion further, we want to specify the limitations of the previous work based on the true meaning of insider attack detection: detecting insider behavioral anomalies.

(1) All previous research focuses on profiling the signature of a user’s queries or transactions [Kamra et al. 2008, Chagarlamudi et al. 2009; Liu and Huang. 2009; Mathew et al. 2010; Raissi-Dehkordi and Carr. 2011; Rathod et al. 2012]; however, as Greitzer mentions, we need to focus on an insider’s behavioral factors to support insider threat analysis [Greitzer and Hohimer. 2011]. From this perspective, the query or transaction anomaly detection mentioned in previous work would be a subset of insider attack detection, but not the whole of it.

(2) A query statement preprocessing algorithm on a query statement takes on a critical role in detecting query or transaction pattern anomalies, but, to the best of our knowledge, no query statement preprocessing algorithm can perfectly identify a query statement from a holistic perspective. Accordingly, no query statement preprocessing algorithm is able to perfectly detect query or transaction differences. For example, two queries which are interpreted as the same by the same query statement preprocessing algorithm could do different work in the real world [Mathew et al. 2010].

(3) Previous approaches are not suitable for a dynamic database whose internal state is constantly changing. Mathew mentioned that data insertion is not permitted in the data-centric approach [Mathew et al. 2010]. Syntax-centric approaches did not mention this issue, but we assume that a valid set of data and user profiles should be updated every time a new valid access happens, because, if not, it will generate a high ratio of false positives. The time complexity of profiling a calculation would be non-trivial while the size of data grows bigger.

5. PROBLEM STATEMENT
From the limitations mentioned above, we deduced the three problem statements we need to solve:

(1) Responding to the first limitation, what data preprocessing methodology is available to not only detect an anomalous query or transaction, but also monitor the insider’s behavioral pattern which can be used for further analysis if we collect enough data using the methodology? If we define a unit of insider behavior as a probability that the insider would execute the query and the corresponding time the query is executed, we would be able to represent the insider’s behavioral pattern with a series of units of the insider’s behavior.

(2) Responding to the second limitation, what approach is available to observe a query or transaction in multiple perspectives? Simply, we could use multiple different query statement preprocessing algorithms to observe a query statement from different perspectives. Then, if we use the differently preprocessed queries from the different query statement algorithms for the data preprocessing methodology in (1), we would detect anomalous behavior which is not able to be detected using only one query statement preprocessing algorithm.
Responding to the third limitation, what approach is suitable to detect an anomalous query or transaction for a dynamic database? Machine learning would not be a good approach because it needs a training phase, and the time spent in training phase would non-trivial, especially if the system deals with thousands of audit logs a minute and the amount of training data is huge. Under those circumstances, it would not be possible to monitor insider behavior in real time. The query probability based approach in (1) could be accomplished without machine learning.

6. OBJECTIVES
We propose a novel approach based on the problem statement above. In summary, the purpose of the system is to not only detect an anomalous query executed by an insider, but also collect the insider’s behavioral data cross three main vectors: query transition probability, query probability and execution time. Also we provide an architecture of a system to collect the data; however, this would still be far from detecting an insider attack because, as Greitzer mentioned, the very required step to detect insider attack is to collect sufficient real world data that enables adequate scientific verification and validation of proposed solutions for insider attack detection [Greitzer and Hohimer. 2011]. Briefly, our approach answers what data we need to collect to detect an insider’s abnormal behavior and how to do so.

We call the implementation of our approach the Multiple-Criteria Query Probabilities Based Database Insider Attack Monitoring System. The “multiple-criteria” means that multiple query statement preprocessing algorithms are applied to monitor the insider’s behavior in multiple perspectives.

We deliver below objectives for this paper as presentation of.

- Multi-Preprocessing Algorithms
  - Operation Related Column Name (ORCN) based query statement preprocessing algorithm.
  - Query Command and Table Name (QCTN) based query statement preprocessing algorithm
- Query Probabilities Based Database Insider Monitoring Methodology based on Markov Mathematical Model
  - Query Transition Probability
  - Query Probability
  - Probabilities based Query Log
- Query Probabilities Time Series Graph
- Multi-Criteria Query Probabilities Based Insider Attack Monitoring System.

Moreover, since the proposed system would need relatively higher computing power and larger storage to store data, than the systems mentioned in previous work, we also propose an architecture for the monitoring system which provides the following features:

- Capability to process large amount of log data to provide real-time monitoring result,
- Cloud based to distribute the workload of log processing and monitoring over multiple nodes,
7. OVERVIEW OF ARCHITECTURE OF MULTIPLE-CRITERIA QUERY PROBABILITIES BASED DATABASE INSIDER ATTACK MONITORING SYSTEM

Figure 1 shows the Physical level architecture of the Multiple-Criteria Query Probabilities Based Database Insider Attack Monitoring System using a Unified Modeling Language (UML) deploy diagram. The system consists of five nodes, and all the nodes communicate using JavaScript Object Notation (JSON) through the HTTP protocol. The DB Log Sender node sends MySQL DBMS audit logs to the Preprocessor node. The Preprocessor node preprocesses the database audit logs to change them into two forms: Extended ORCN-Based Queries and Extended QCTN-Based Queries, and then sends them to the Logger node. The Logger node updates several tables in the Logger database and stores the preprocessed queries with the Probabilities Based Query Log forms. The Query Probability Calculator node periodically calculates query probabilities using the Query Transition Tables and a Markov Chain mathematical model for each user. The Monitor node renders the Query Probabilities Time Series Graph with the stored Probabilities Based Query Logs to visualize the behavior of insiders in real time.

8. DB LOG SENDER NODE

The DB Log Sender node simply sends a list of auditing logs from the audit log table in the database and flushes the table periodically. We used MySQL database for the monitored database. The General Query Log table was used to audit the insider’s queries. We determined the unit being monitored as a query, not a transaction, since we want to handle ad-hoc queries and also monitor at a fine-grained query level. Figure 2 is the schema of the General Query Log table named general_log. The event_time is the query execution time. The user_host is a combined string of the user name who executed the query and host name where the query was executed. The
thread_id is an ID of the thread by which the query was executed. The server_id is a server ID. The command_type tells the category of the insider’s request, for example: connect, query or etc. The argument is a SQL statement the user executed. We decided to focus on queries, who executed, where and when executed. Accordingly, we selected event_time, user_host and argument attributes to monitor insider behavior.

![Fig. 2. The General Query Log Table.](image)

9. PREPROCESSOR NODE

9.1 Approach

The Preprocessor node first interprets an incoming general log from the DB Log Sender node in multiple perspectives with multiple query statement preprocessing algorithms, then sends them to the Logger node. We introduce two query statement preprocessing algorithms: Query Command and Table Name (QCTN) and Operation Related Column Name (ORCN) based query statement preprocessing algorithms.

9.1.1. Query Command and Table Name (QCTN) Based Preprocessing Algorithm. The QCTN algorithm was used by Rathod to redefine a query: a query is identified by the query command and table names [Rathod et al. 2012]. The paper did not specifically mention the case that multiple nested FROMs are used in a query, so we decided not to count the table name used in the nested FROMs.

9.1.2. Operation Related Column Name (ORCN) Based query preprocessing algorithm. The QCTN based algorithm could not detect the differences in WHERE. Accordingly, we need another algorithm to support the shortage of the QCTN based algorithm. We came up with a new query statement preprocessing algorithm to identify a query by the column names appeared in the WHERE. The table name, concatenated with the column name, is also considered to identify the column name. Table I shows a query statement interpreted by the two query statement preprocessing algorithms.

| Query Statement | SELECT table_a.column_b, table_b.column_b FROM table_a, table_b WHERE table_a.column_c = table_b.column_d |
| Query Command and Table Name (QCTN) Based | qctn#Select(table_a&table_b) |
| Operation Related Column Name (ORCN) Based | orcn#(table_a.column_c&table_b.column_d) |

9.2 Extended QCTN Based and Extended ORCN Based Queries

Figure 3 shows the attributes of the Extended QCTN Based Query and Extended ORCN Based Query object which are the representations of the preprocessed queries by the QCTN and ORCN query preprocess algorithms. Both have six extra attributes to process the objects in the Logger node (Table II).
Table II. The Six Extra Attributes Added to the Extended QCTN and ORCN Based Query Objects.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>A string representing the algorithm, &quot;qctn&quot; or &quot;orcn&quot;</td>
</tr>
<tr>
<td>hostname</td>
<td>An address of a computer the query was executed</td>
</tr>
<tr>
<td>id</td>
<td>A preprocessed query ID</td>
</tr>
<tr>
<td>rawStatement</td>
<td>The original query statement in the audit log from the DB Log Sender node</td>
</tr>
<tr>
<td>timestamp</td>
<td>A timestamp when the query executed</td>
</tr>
<tr>
<td>userName</td>
<td>A user name who executed the query</td>
</tr>
</tbody>
</table>

The id in Table II represents a preprocessed query ID which can uniquely define the preprocessed query log. The representations of a query in Table I are used for the IDs for the two types of preprocessed queries.

10. LOGGER NODE

10.1 Approach

The preprocessed queries are received by the Logger node. The role of this node is to receive the preprocessed queries, then update the Query Transition Probability Tables for all the preprocessed queries executed by each user, then store the Probability Based Query Logs for each user. Figure 4 shows the attributes in the Probability Based Query Log. It is basically an extension of a preprocessed query with additional two attributes: the Query Probability and Query Transition Probability.
Probabilities for each user are dynamically updated while the user is interacting with the monitored database. Consequently, the Query Transition Probability not only detects an anomalous query but also shows how frequently each query is executed in query transition view. The calculation of the Query Transition Probability is pretty simple and straightforward. For this calculation, we need three types of tables: the User Table to keep tracking each user's previous query ID, the Transition Table Name Index Table for each user to keep track of all the queries the user has executed, and multiple Query Transition Tables for each query listed in the Transition Table Name Index Table for the user are required to keep tracking all the query transitions. Algorithm 1 shows the steps to update the Transition Table Name Index Table and Query Transition Table to calculate the Query Transition Probability. Algorithm 2 is the steps to calculate Query Transition Probability. The steps in Algorithm 2 happen after the update in Algorithm 1 is completed.

---

**Algorithm 1.** The Query Transition Index and Query Transition Table Update

**Input:** A list of preprocessed queries \( (Q_1 \ldots Q_n) \) preprocessed by the same query statement preprocessing algorithm.

**Output:** None

for each query \( Q_i \) in the list of preprocessed queries \( (Q_1 \ldots Q_n) \) do

user\(_i\) = get user name from \( Q_i \);

TransitionTableName\(_i\) = a Transition Table Name Index Table for user\(_i\);

if TransitionTableName\(_i\) is not in the Transition Table Name Index Table then

TransitionTableName\(_i\) = Random(UUID) in string form for the Query Transition Table name;

create a Query Transition Table with TransitionTableName\(_i\);

insert \( Q_i \) with TransitionTableName\(_i\) into QueryTransitionTable\(_{user_i}\);

end

PrevQuery\(_{user_i}\) = get a previous query user\(_i\) executed from the User Table;

QueryID\(_i\) = a query ID from \( Q_i \);

if PrevQuery\(_{user_i}\) is not NULL then

QueryTransitionTable\(_{PrevQuery\(_{user_i}\)}\) = a Query Transition Table of PrevQuery\(_{user_i}\);

if QueryID\(_i\) does not exists in QueryTransitionTable\(_{PrevQuery\(_{user_i}\)}\) then

insert QueryID\(_i\) into QueryTransitionTable\(_{PrevQuery\(_{user_i}\)}\);

end

increment the count column of QueryID\(_i\) by one;

end

update the User Table with user\(_i\) and QueryID\(_i\);

end

---

**Algorithm 2.** The Query Transition Probability Calculation

**Input:** A preprocessed query \( (Q_k) \) preprocessed by a query statement preprocessing algorithm.

**Output:** A query transition probability \( \text{QueryTransitionProbability}_k \).

QueryTransitionProbability\(_k\) = 0.0;

user\(_k\) = get user name from \( Q_k \);

QueryID\(_k\) = a query ID from \( Q_k \);

PrevQuery\(_{user_k}\) = get a previous query user\(_k\) executed from the User Table;
if $\text{PrevQuery}_{user_k}$ is not NULL
then

\[
\text{QueryTransitionTable}_{\text{PrevQuery}_{user_k}} = \text{a Query Transition Table of } \text{PrevQuery}_{user_k};
\]
\[
\text{CountSum} = \text{the sum of all count columns in } \text{QueryTransitionTable}_{\text{PrevQuery}_{user_k}};
\]
\[
\text{QueryTransitionProbability}_{k} = \text{the count of } \text{QueryID}_k / \text{CountSum} ;
\]

end

10.2 Cassandra DB

We used the Cassandra database to implement the system database. Cassandra is a NoSQL database that can store and process big amounts of data efficiently because of high scalability [Abramova and Bernardino, 2013]. Wei proved the effectiveness of using MongoDB, which is one of the NoSQL databases for huge amounts of data monitoring [Wei et al. 2011]; however, the performance of Cassandra, especially for writing related work, surpasses the performance of MongoDB [Abramova and Bernardino, 2013]. We decided to use Cassandra over MongoDB because logging and monitoring is writing intensive work. Figure 5 shows the schema of all the tables related to the Query Transition Probability Calculation mentioned in Algorithms 1 and 2.

To increase performance on implementation, we created a table for each Query Transition Table which keeps tracking the sum of count to reduce calculation time for CountSum in Algorithm 2.
11. QUERY PROBABILITY CALCULATOR NODE

11.1 Approach

Query Probability Calculator node calculates Query Probabilities of all queries each user has executed and updates the Query Probability Table for each user. Query Probability is a query execution probability among all the queries a user has executed. Query Probabilities are calculated based on Query Transition Probabilities which are described in section 10. Then the Markov Mathematical Model is applied to the Query Transition Probabilities to produce Query Probabilities for each user.

11.2 Invariant Property of Markov Chain

The Markov Chain Model is a mathematical system for going through transitions: one state transits to another state on a state space [Norris. 1998]. We built Query Probabilities based on the Query Transition Probabilities with the invariant property in the Markov Chain. I is a countable set called state-space. λ is a distribution: any row vector $\lambda = (\lambda_i : i \in I)$ with total mass $\sum_{i \in I} \lambda_i$ equals 1, non-negative entries and measure on I. Each $i \in I$ is called a state. $P = (p_{ij} : i, j \in I)$ is a stochastic matrix, and every column in P is a distribution [Norris. 1998]. Then we say $\lambda$ is invariant if $\lambda P = \lambda$.

11.3 Query Probability Calculation Using the Invariant Property of Markov Chain

We applied the invariant property to calculate the Query Probability. First we need to define the Query Transition Probability Matrix, K as

$$K = \{ k_{ij} | i, j \in I, n = |I|, \sum_{j=0}^{n} k_j = 0 \}.$$  (1)

$I$ is a set of all the query IDs executed by the same user, and each user has a different state space I. $k_{ij}$ is the Query Transition Probability (Algorithm 2): from query $i$ to query $j$. Every row in K is a distribution, and the sum of the probabilities in each row is 0 (if no j is executed after i) or 1.

Before we construct the Query Transition Probability Matrix using the definition above, we need to consider one variable $p$, the probability that an arbitrary user executes a query related to the previous query. When a user executes a query, the query would be related to the previous query, or it would be an aimless query. The Page rank algorithm by Larry Page motivated us to consider the variable $p$. Page introduces a vector $E$ corresponding to a random surfer periodically jumping to a random web page without following a link [Page et al. 1999]. Similarly, $1 - p$ corresponds to the random user periodically jumping to a random query. This not only prevents the cycle problem while applying the Markov Chain, but also allows us to consider the probability of arbitrary queries. Based on taking $p$ into account, we built the Enhanced Query Transition Probability Matrix M as

$$M = \{ m_{ij} | i, j \in I, n = |I|, \sum_{j=1}^{n} m_j = 1 \ for \ fixed \ i \},$$  (2)

$$r_i = \sum_{j=1}^{n} k_j,$$  (3)

$$\delta = (1 - p)/n,$$  (4)

$$m_{ij} = \begin{cases} pk_{ij} + \delta : r_i = 1 \\ \frac{1}{n} : r_i = 0 \end{cases}.$$  (5)

$r_i$ is the sum of each row in M. $\delta$ represents the probability of executing an arbitrary query j not related to the previous query i.
Then we can get the invariant \( \lambda \) satisfying (8) with a sufficiently large \( n \) and initial distribution (6 and 7), which is a row vector. We define the invariant \( \lambda \) attained from the last equation (8) as the list of Query Probabilities.

\[
X = \{X \mid |X| = |I|, X_i = 1/|I|\}
\]

(6)

\[
\lambda_{\text{init}} = XM
\]

(7)

\[
\lambda M^{(n)} = \lambda
\]

(8)

With all the consideration above, Algorithm 3 shows the steps to calculate the Query Probabilities for all the queries.

**ALGORITHM 3.** The Query Probability Calculation

**Input**: A list of IDs of preprocessed queries by the same algorithm \( (l_1 \ldots l_n) \) a user has executed.

**Output**: The Query Probability List \( (\lambda) \)

\( K = n \times n \) Query Transition Probability Matrix;

*populate* \( K \) with the equation (1);

\( M = n \times n \) Enhanced Query Transition Probability Matrix;

*populate* \( M \) with the equation (3) – (5);

\( \lambda = \) a row vector from (7);

\( \lambda = \) calculate (8);

11.4 Example

To help with understanding, we show how the overall process of the Query Probability calculation operates through a series of figures.

When a query \( Q_1 \) is received, because \( Q_1 \) does not exist in the Transition Table Name Index Table, a new Query Transition Table for \( Q_1 \) is created then \( Q_1 \) is added to the Transition Table Name Index Table (Figure 6).

![Fig. 6. Table Update and Creation with an Unknown Query \( Q_1 \)](image)

When the next query \( Q_2 \) is received, because \( Q_2 \) does not exist in the Transition Table Name Index Table, a new Query Transition Table for \( Q_2 \) is created then the Query Transition Table of \( Q_1 \) is updated with count 1 and probability 1 in the \( Q_2 \) row since there is only one outbound query ID. After all of the above, \( Q_2 \) is added to the Transition Table Name Index Table (Figure 7).
When the next query $Q_3$ is received, because $Q_3$ does not exist in the Transition Table Name Index Table, a new Query Transition Table for $Q_3$ is created then the Query Transition Table of $Q_2$ is updated with count 1 and probability 1 on $Q_3$ row since there is only one outbound query ID. After all of the above, $Q_3$ is added to the Transition Table Name Index Table (Figure 8).

When the next query $Q_1$ is received, because $Q_1$ exists in the Transition Table Name Index Table, only Query Transition Table of $Q_3$ is updated with count 1 and probability 1 on $Q_3$ (Figure 9).
When the next query $Q_4$ is received, because $Q_4$ does not exist in the Transition Table Name Index Table, a new Query Transition Table for $Q_4$ is created then the Query Transition Table of $Q_1$ is updated with count 1 and probability 0.5 on both rows $Q_2$ and $Q_4$ since there is two outbound query IDs. After all of the above, $Q_4$ is added to the Transition Table Name Index Table (Figure 10). Figure 11 represents the final states of the Query Transition Tables of $Q_1$ – $Q_4$ and the Transition Table Name Index Table. Finally, Figure 12 and 13 shows the Enhanced Query Transition Probability Matrix calculated by Algorithm 3.
Fig. 11. Final States of the Query Transition and Transition Table Name Index Tables.

<table>
<thead>
<tr>
<th>Pattern_ID</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_2</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>Q_4</td>
<td>1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern_ID</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q_4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 12. The Enhanced Query Transition Probability Matrix Calculated by Algorithm 3.

<table>
<thead>
<tr>
<th>Q_1</th>
<th>Q_2</th>
<th>Q_3</th>
<th>Q_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>p × 0 + δ</td>
<td>p × 0 + δ</td>
<td>p × 0 + δ</td>
<td>p × 0.5 + δ</td>
</tr>
<tr>
<td>p × 0 + δ</td>
<td>p × 0 + δ</td>
<td>p × 1 + δ</td>
<td>p × 0 + δ</td>
</tr>
<tr>
<td>δ × 1/4</td>
<td>p × 0 + δ</td>
<td>p × 0 + δ</td>
<td>p × 0 + δ</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q_1</th>
<th>Q_2</th>
<th>Q_3</th>
<th>Q_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0375</td>
<td>0.4625</td>
<td>0.0375</td>
<td>0.4625</td>
</tr>
<tr>
<td>0.0375</td>
<td>0.0375</td>
<td>0.8875</td>
<td>0.0375</td>
</tr>
<tr>
<td>0.8875</td>
<td>0.0375</td>
<td>0.0375</td>
<td>0.0375</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Fig. 13. The Query Probability Calculated by Algorithm 3.

<table>
<thead>
<tr>
<th>Query Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_1 0.3078</td>
</tr>
<tr>
<td>Q_2 0.2138</td>
</tr>
<tr>
<td>Q_3 0.2640</td>
</tr>
<tr>
<td>Q_4 0.2138</td>
</tr>
</tbody>
</table>

12. MONITOR NODE

12.1 Approach

Monitor node monitors insider behavior by inspecting the Probability Based Query (PBQ) logs of a user. The purpose of this node is to detect anomalous behavior of a user and predict insider attack. In the Multiple-Criteria Query Probabilities Based Database Insider Attack Monitoring System, logging and monitoring are done by different nodes for two reasons. Firstly, there is no reason to combine the two tasks in one program since they are logically different tasks. Secondly, the Monitor node
has architecturally moved most of the monitoring workload to the client side: the Monitor node is only sending the PBQ logs to a client, and analysis with any calculation is done by the client.

We propose a simple method to monitor and predict insider attack, Total-Mean and Last-k-Mean Based Insider Attack Monitoring. Actually, any monitoring method can be used for the Monitor node because this monitoring method element is totally on the client side since the server is just periodically sending log data to the client. The Total-Mean and Last-k-Mean Based Insider Attack Monitoring consists of three steps: calculation, presentation and interpretation.

12.2 Calculation
Based on the two parameters we already have for monitoring insider behavior: the Query Transition Probability and Query Probability, we add four more simple statistical parameters to predict insider attack: mean and last-k-mean for each Query Transition Probability ($P$) and Query Probability ($Q$).

$$\frac{1}{n} \sum_{i=0}^{n} P_i, \quad \text{(1)}$$

$$\frac{1}{n} \sum_{i=0}^{n} Q_i, \quad \text{(2)}$$

$$\frac{1}{k} \sum_{i=n+1-k}^{n} P_i, \quad \text{(3)}$$

$$\frac{1}{k} \sum_{i=n+1-k}^{n} Q_i. \quad \text{(4)}$$

$i$ is PBQ log, $n$ is the total number of PBQ logs, $k$ is an arbitrary number for the last $k$ PBQ queries. (1) is the mean of the Query Transition Probability, (2) is the mean of the Query Probability, (3) is the last-$k$-mean of the Query Transition Probability, (4) is the last-$k$-mean of the Query Probability. The $k$ will not be too small or large because the mean should not be changing too drastically or stiffly, but gracefully in an elastic manner.

12.3 Presentation
We used the Query Probabilities Time Series Graph to render the four parameters (1 – 4) in a time series. The purpose of this graph is to visualize the insider’s behavior, which can be easily readable by a human.

12.4 Interpretation
We set the two time series lines, mean of the Query Transition Probabilities and Query Probabilities (1, 2), as representation of standard lines. Normally, we expect that time series of the last-$k$-mean of the Query Transition Probability and last-$k$-mean of the Query Probability (3, 4), active lines would fluctuate around the standard lines because the insider would not execute high probability queries every time. However, if the two active lines keep staying at 3 or 4 categories in Figure 6 for a relatively long time, we assume that it is anomalous behavior. In Figure 6, category 3 is more suspicious than 2 because category 2 has a higher transition probability than 3, which can be considered more normal.
Fig. 6. The Query Probabilities Time Series Graph

13. EVALUATION
13.1 Approach

We monitored user behavior in a MySQL database having Figure 7 schema. We fed the seven queries in Table III with the Feeding order in Figure 8. In Figure 8, probability of the edge from query ID 7 to 1 is 1.0, other edges are randomly selected. We used the Random class in JAVA to achieve randomness from query to query. We iterate the set of random queries, in Figure 8, 500 times in the feeding phase. We assume that normal queries are executed during the feeding phase.

Table III. Feeding Queries.

<table>
<thead>
<tr>
<th>ID</th>
<th>Query Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT * FROM Buildings WHERE building_name='Dougan';</td>
</tr>
<tr>
<td>2</td>
<td>SELECT * FROM DeliveryStatus WHERE demand_id=1;</td>
</tr>
<tr>
<td>3</td>
<td>SELECT * FROM DemandHistory WHERE demand_quantity=3;</td>
</tr>
</tbody>
</table>
SELECT * FROM ItemMaxAllotmentForRoom WHERE item_max_allotment=11;
SELECT * FROM Items WHERE total_quantity_on_hand=100;
SELECT * FROM ItemTypes WHERE type_name='Chairs';
SELECT * FROM Rooms WHERE building_id='CP';

Fig. 8. The Order of Query Feed.

After the feeding phase, we executed a query which was not in the feeding queries in Table III as below, and the result is Figure 9.

INSERT INTO dod.Items (item_id, item_name, item_type, total_quantity_on_hand)
VALUES (1, 'desk', 1, 1).

Fig. 9. Anomalous Query Detected by Both Pre-processing Algorithms.

Both the QCTN and ORCN query preprocessing algorithms successfully detect the anomalous query, which means that the query has not been executed by the user. We easily notice that the query never has been executed by observing the Query Probability, 0.0. Another query we executed was that the query can only be distinguished by the ORCN query preprocessing algorithm. The below query has query command “SELECT” and table name “DemandHistory” which are used to build the QCTN based query, and the query signature is already registered by the query ID 3 in Table III. As expected, Figure 10 shows that the below query is detected as anomalous only by the ORCN preprocessing algorithm, the Query Probability 0.0.
SELECT * FROM DemandHistory WHERE item_id=1.

From the experiment above, we showed how the proposed insider attack monitoring system detects an anomalous query. Then we also created experiments to show how the system detects anomalous behavior: specifically a set of trusted queries, but in different order. Figure 11 is the order of query IDs from 7 to 1. Each query in Figure 11 is the same as each query in Table III, but the whole ordered set is anomalous because the user has never executed the queries in the same order. Accordingly, we can assume that the user’s account might be compromised, or that the user performs a totally different task from previously, which might be suspicious.

We randomly executed the set of ordered queries in Figure 11 two times while the series of queries are executed in Figure 8, one query every second. Then we observed the result through the Query Probabilities Time Series Graph—the two red boxes in the Query Probabilities Time Series Graphs with different combinations of variables p mentioned in the chapter 11 and k mentioned in the chapter 12 (Figure 12 to 23).

Fig. 11. Anomalous Query Detected by Both Pre-processing Algorithms.
Fig. 12. Detecting Anomalous Behavior with $p=0.65$ and $k=5$.

Fig. 13. Detecting Anomalous Behavior with $p=0.65$ and $k=10$.

Fig. 14. Detecting Anomalous Behavior with $p=0.65$ and $k=15$. 
Fig. 15. Detecting Anomalous Behavior with $p=0.75$ and $k=5$.

Fig. 16. Detecting Anomalous Behavior with $p=0.75$ and $k=10$.

Fig. 17. Detecting Anomalous Behavior with $p=0.75$ and $k=15$. 
Fig. 18. Detecting Anomalous Behavior with $p=0.85$ and $k=5$.

Fig. 19. Detecting Anomalous Behavior with $p=0.85$ and $k=10$.

Fig. 20. Detecting Anomalous Behavior with $p=0.85$ and $k=15$. 
Fig. 21. Detecting Anomalous Behavior with $p=0.95$ and $k=5$.

Fig. 22. Detecting Anomalous Behavior with $p=0.95$ and $k=10$.

Fig. 23. Detecting Anomalous Behavior with $p=0.95$ and $k=15$. 
From observations of Figure 12 through 23, we can easily see that Query Transition Probability is better related to representing insider behavior rather than Query Probability. It is obvious because Query Probability is not related to the order of queries, but the rank of queries, how frequently a query is executed in general. Accordingly, the variable p, which is used for the Query Probability calculation does not have an important role in detecting behavior anomalies; however, the variable k turned out to be important. From the pairs of Figure 12 – 14, 15 – 17, 18 – 20, 21 – 23, we realize that if k is larger than 15, the slope would be too gentle to detect anomalous behavior. Conversely, if k is smaller than 5, the graph will be too noisy to detect anomalous behavior. The gentleness of the slope starts appearing from when k is 10 in Figure 16 second red box. In conclusion, the ideal k value would be between 5 and 10.

14. CONCLUSION

The proposed system has four benefits. Firstly, it can detect a query anomaly by using Query Probability. Secondly, the Query Transition Probability is also able to show insider behavioral anomaly. Thirdly, the system is more sensitive to detecting anomalies by applying a multiple query statement preprocessing algorithm than a system using just one query statement preprocessing algorithm. However, as we repeated mention in this paper, our approach is still far from real insider attack detection since our approach is not to detect insider attack but to collect data which would be useful for data mining to create an anomalous insider behavior profile. The three findings mentioned above are meaningful to setting up a framework to collect insider behavioral data.

REFERENCES


