COMPLEX EVENT PROCESSING FOR SENSOR BASED DATA AUDITING

Christian Lettner, Christian Hawel, Thomas Steinmaurer and Dirk Draheim
Software Competence Center Hagenberg (SCCH), Softwarepark 21, A-4232 Hagenberg, Austria

Keywords: Business Processes, Complex Event Processing, Data Auditing, Multi-tier Architecture.

Abstract: Current legislation demands organizations to responsibly manage sensitive data. To achieve compliance, data auditing must be implemented in information systems. In this paper we propose a data auditing architecture that creates data audit reports out of simple audit events at the technical level. We use complex event processing (CEP) technology to obtain composed audit events out of simple audit events. In two scenarios we show how complex audit events can be built for business processes and application users, when one database user is shared between many application users, as found in multi-tier architectures.

1 INTRODUCTION

1.1 Data Auditing and Regulatory Compliance

Data auditing aims to capture the activities that are performed on data (Natan, R., 2005). Data can be accessed, inserted, modified or deleted. According to the requirements, data auditing can be implemented on different levels of granularity. For data stored in database systems an approach found very often is to store the old value of a data field that has been modified. In addition, the timestamp and the user who performed the modification are stored. Starting from this simple approach, data auditing can be extended to be able to provide a snapshot of the data for an arbitrary time in the past. Additionally, all activities performed on the data are captured and stored. Such a complete audit trail can lead to massive data volume which is very hard to handle. Therefore, each data auditing solution poses a trade-off between granularity of audit data and available of storage. Data auditing includes also read-only operations. This is of particular importance, because not all commercial database systems provide built-in solutions for that.

In recent years, data auditing has gained wide attention as new legislation demands organization to carefully manage their sensitive data (Johnson, C., Agrawal, R., 2006). In the US, the Sarbanes-Oxley Act (SOX) of 2002 regulates the requirements for corporate financial data. Senior executives have to take individual responsibility for the accuracy and completeness of corporate financial reports. The health care system in the US is regulated by the Health Insurance Portability and Accountability Act (HIPAA) of 1996. In the EU, the European Union Privacy Directive of 1998 regulates the management of sensitive privacy data in general.

To ensure regulatory compliance, organizations must run appropriate information systems. One core requirement for these systems is data auditing. The systems must be able to keep track of read-only and modification operations to sensitive data. Also, these activities must be assigned to unambiguous business users in charge. Most organizations use multiple information systems to perform their daily business. It's very likely that sensitive data is scattered across different applications and databases. Isolated data audits on single databases are not sufficient for compliance. To get a complete picture of all activities on sensitive data, an enterprise-wide data auditing solution across data management systems is mandatory.

1.2 Data Auditing Requirements

Several requirements on data auditing solutions can be derived based on these regulatory frameworks:

- *Enterprise-wide solution*. Organizations use many different applications to operate their business. An enterprise-wide solution for data auditing is needed to get a complete overview

Lettner C., Hawel C., Steinmaurer T. and Draheim D. (2008).
COMPLEX EVENT PROCESSING FOR SENSOR BASED DATA AUDITING.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 485-491
DOI: 10.5220/0001707004850491

of the activities performed on the data from different applications and databases.

- *Audit of data manipulation operations*. Especially for sensitive data, data audit solutions must be able to keep track of any data changes in a thorough change log.
- *Audit of read-only operations*. Another common requirement is to provide an audit trail of all business users who have accessed certain data entities.
- *Unambiguous application user identification*. For meaningful data audits, the application user who performed an operation must be identified unambiguously. This is especially challenging for data audits performed within the database management system, when multiple application users share the same database user.
- *Audit of privileged user and separation of duties*. Privileged users are able to perform more operations and may have different access patterns on data entities than business users, e.g. a system administrator is able to connect to a database through a development tool, actually bypassing the business application used by business users and issue ad hoc queries. The audit system must provide a mechanism to audit such activities performed by system administrators. Further, the data audit system must not be run by the system administrator, to achieve separation of duties.
- *Affected sensitive data entity*. Some regulations require it to identify the entities affected by the activities performed on the data, e.g. the customer, whose address has been updated must be identified.

Other requirements basically stem from organizational or technical needs:

- *Platform independency*. If an organization uses different platforms for their applications, the audit solution should be able to run on all of them.
- *Transparent to new application releases*. New application releases should not break the audit solution. Only minor changes to the audit solutions should be necessary.
- *Audit at the level of business processes*. Significant data audit trails must be at the business process level. The audit entry 'Employee X has changed the address of Customer Y' is much more meaningful than a series of *Select* and *Update* statements.
- *No impact on application performance*. The influence on application performance should be kept as low as possible.
- *Tools to analyze and manage audit data*. The audit solution must provide appropriate tools to analyze the audit data. If auditing is performed at high granularity, tools that manage the huge amount of audit data must be provided as well.
- *Selective data auditing*. It should be possible to enable/disable data audits on certain data entities.

1.3 Complex Event Processing (CEP)

Compared to the every day usage of 'event' as 'something that happens', in CEP an event is an object (Luckham, D., 2005). The event object describes the activity and is possibly related to other event objects. The three most common and important relationships between events are:

- *Time*. Time is a relationship that orders events. For example event *A* happened before event *B*.
- *Cause*. If event A had to happen in order for event *B* to happen, then *A* caused *B*.
- *Aggregation*. If event *A* consists of events B_1, B_2, B_3, …, then *A* is an aggregation of all the events B_i. Conversely, the events B_i, are members of *A*. Aggregation is an abstraction relationship.

Complex event processing employs techniques as detection of complex event patterns for many events, event correlation, event abstraction and event hierarchies. Typical application domains of CEP are financial trading systems where a lot of events must be processed at very high speed.

1.4 Related Work

(Agrawal, R., Bayardo, R., Faloutsos, C., Kiernan, J., Srikant, R., 2004) presents an auditing framework for queries on sensitive data entities. The sensitive data entities must be specified using audit expressions. The queries and audit expressions are combined and transformed to an audit query, which will be executing against a backlog database (represents the state of the database when the query was executed). The audit query evaluates all data rows processed by the original query and determines whether the query has accessed sensitive data specified by the audit expression. The advantage of this approach is that queries are also audited if the information disclosed by the query is not part of the output. As the pool of suspicious queries identified by the auditing framework can become very large, in

(Agrawal, R., Evfimievski, A., Velu, R., 2007) an approach to rank suspicious queries is presented. Three different measures are provided to measure proximity there. (Motwani, R., Nabar, S., Thomas, D.) extends this work and provides a formal foundation to audit a batch of SQL queries.

(Chen, S., Jeng, J., Chang, H., 2006) uses CEP for business performance management. The focus is on an extension to the Zurich Correlation Engine to allow structural events in XML format to be processed as well. (Lee, W., Fan, W., 2001) employs data mining algorithms for intrusion detection systems. Data mining is used to react to different attack patterns. For data auditing, the access pattern is always the same, thus eliminating the need for data mining algorithms.

The proposed data auditing architecture in this paper is also suitable for the business activity monitoring approach presented in (Mangisengi, O., Pichler, M., Auer, D., Draheim, D., Rumetshofer, H., 2007).

The remainder of this paper is organized as follows: Section 2 proposes an architecture for an enterprise-wide audit solution. It presents a categorization of locations – so called audit hotspots – where audits can be performed. A technical introduction to auditing based on complex event processing based on a simple example is provided in Section 3. Section 4 describes two audit scenarios for the proposed auditing architecture. Finally, a conclusion and future work are presented in Section 5.

2 DATA AUDITING ARCHITECTURE

We use a sensor based data auditing architecture. Sensors are lightweight agents whose task is to listen on simple audit events, for example a query sent to a database management system. The sensors are scattered across the enterprise and communicate detected audit events to one central audit event processing system. After the events are processed, the audit trail is forwarded to the central reporting system.

Usually, it is very likely that different applications to be audited require different types of sensors. For example you could think of a sensor for the file server to get access information to files that contain sensitive information. We call the location at which, a sensor is installed an audit hotspot. Figure 1 shows a categorization of possible audit hotspots for an information system that implements a three-tier architecture:

- *Audit hotspot 1*. Auditing is implemented within the client application. This is the only approach where the data can be audited that actually has been seen by the user, e.g., in a GUI driven application an audit event is only created when the user actually switches to the area containing sensitive information. The main disadvantage of this approach is that it requires to be implemented for each client application type, so it is not a generic solution.
- *Audit hotspot 1*. Auditing is implemented within the client application. This is the only approach where the data can be audited that actually has been seen by the user, e.g., in a GUI driven application an audit event is only created when the user actually switches to the area containing sensitive information. The main disadvantage of this approach is that it requires to be implemented for each client application type, so it is not a generic solution.
- *Audit hotspot 2a*. Traffic between the client and the server is scanned at the network level. This generic solution provides an audit solution for all client applications communicating to the application server. It is non-intrusive, which means, there is no influence on application performance. The challenge in this approach comes with the analysis and interpretation of the audit data. Direct access to data, bypassing the application, will not be audited (for example: privileged users that use a development tool).
- *Audit hotspot 2b*. Communication between the client and the server is intercepted by the sensor that is implemented as a proxy application. This approach is similar to the previous one, except that the whole client server communication runs through the sensor. This allows the sensor to react on malicious activities on data. It is much like a data firewall for intrusion detection systems and is able to enforce privacy policies.
- *Audit hotspot 3a*. Auditing is implemented within the application server. Built-in features of the application server are used. This approach represents a generic solution for all client applications using the application server. On the other side, generic built-in auditing features of the application server may not provide the required audit granularity.
- *Audit hotspot 3b*. Services are extended to perform the audit. This intrusive approach requires that all services have the auditing

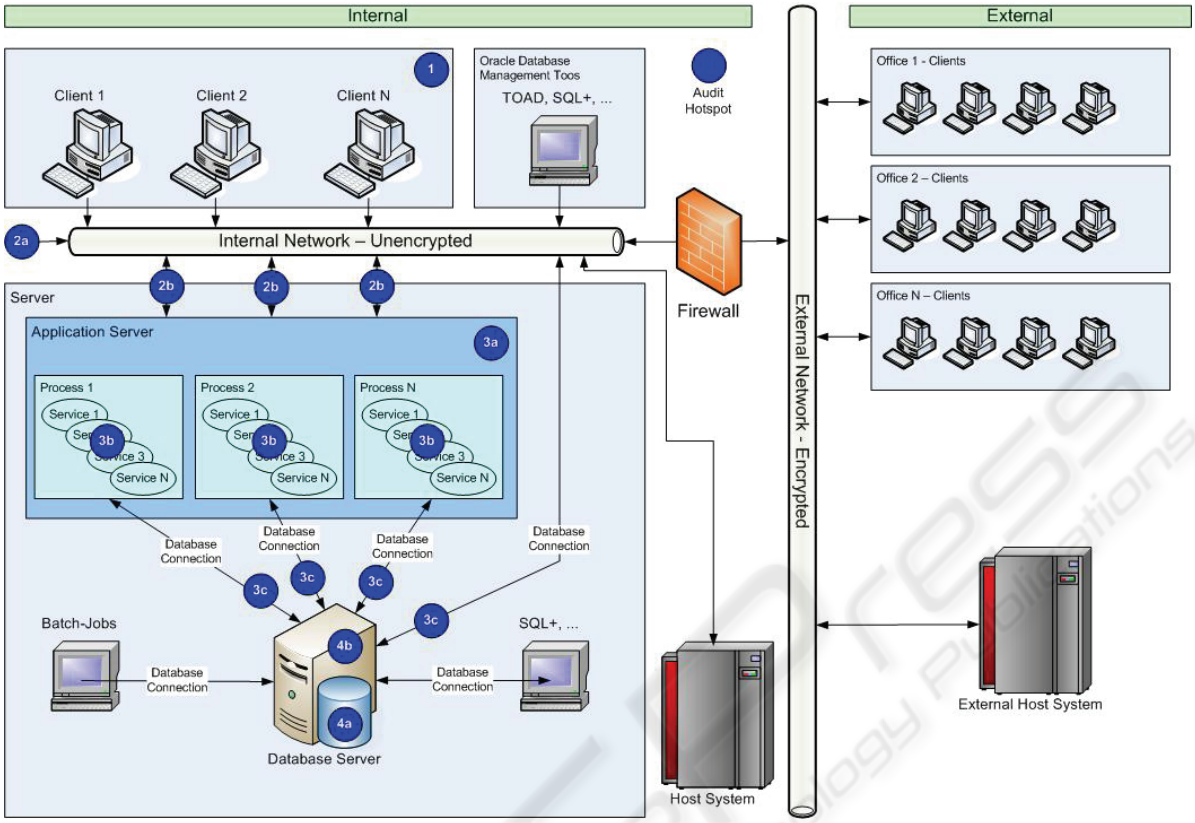

Figure 1: Audit-hotspot in a three-tier architecture.

functionality implemented. On the other side, all data sent to and from the client is available.

- *Audit hotspot 3c*. Communication between the server and the database is scanned at the network level. This approach is very similar to audit hotspot 2a which scans the communication between the client and the application server. The difference is in the recorded audit event types. In this approach, only the SQL statements issued by the application server are audited. This makes it even more difficult to interpret and analyze the detected audit event. We utilize complex event processing technology to perform this task. A disadvantage of this approach is that data cached by the application server that is sent to the client will not be audited. On the other hand, this approach is also non-intrusive and provides a generic solution. It even audits activities on data that bypasses the application server. If there is the requirement to audit privileged users that use development tools and are able to issue ad hoc queries, then performing auditing at this audit hotspot is one way.
- *Audit hotspot 4*. Auditing is implemented within the database management system. This approach provides similar audit information as audit hotspot 3c. Built-in database features can be used to setup and maintain an audit mechanism, with low additional costs. On the other hand, it is an intrusive approach and influences application performance at the server. To implement audits of read-only operations, this feature must be explicitly supported by the database management system.

Choosing the appropriate audit hotspot in a given scenario mainly depends on the audit information available at a particular audit hotspot. Sometimes it is reasonable to extend the application in a way, to make the required information available, e.g., in a connection pooling environment, the data access layer could be enriched with application user information). Often, a combination of audit hotspots will be necessary to leverage an enterprise-wide

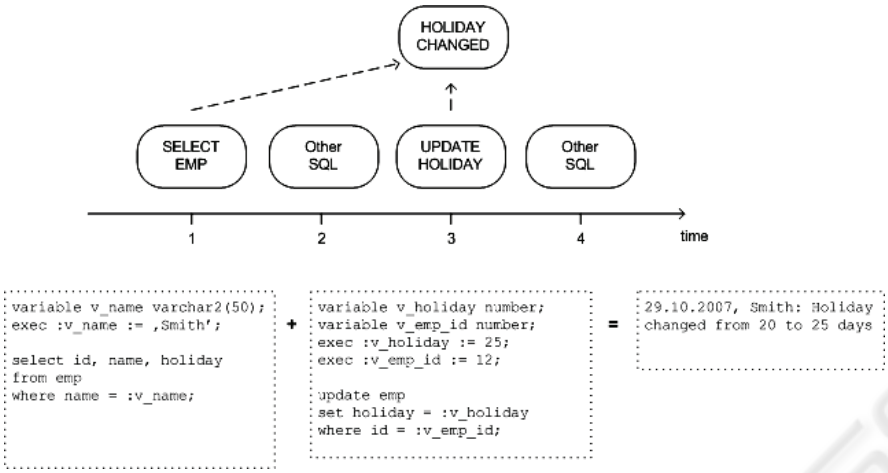

Figure 2: Event processing for change log.

auditing solution.

In our architecture, data audit events created at the audit hotspots are communicated to one central event processing system. The event processing system must be able to detect patterns within the audit event stream. The next section shows a simple example on how this can be accomplished using complex event processing technology. Then, the results from the event processing system are stored in a data warehouse for further analysis.

3 SIMPLE EXAMPLE FOR DATA AUDITING USING CEP

In this section we present a simple example how complex event processing can be used for data auditing. Based on a sensor that scans the communication between the application and database server (audit hotspot 3c), simple *Select* and *Update* statements are audited. Using complex event processing, a change log for the attribute *holiday* of the table *emp* will be generated. Every time the attribute *holiday* of an employee is changed, a single complex audit event will be generated. Figure 2 shows how the events are processed.

The first statement executes a *Select* on table *emp* for the employee 'Smith'. The tuple *(12, Smith, 25)* will be returned. Two statements later, an *Update* on the attribute *holiday* is performed for the same employee (*id = 12*). All the statements make use of parameterized queries. Due to the fact that this two statements happen within a defined time span (here at most 2 seconds), the statements will be correlated by the event engine, which generates a new complex event for the change. The change event is composed of the attributes *time*, *employee name*, old/new value of attribute *holiday*, where:

- the *time* equals the execution time of the *Update* statement,
- the *employee name* and the old value of *holiday* are received from the result tuple of the *Select* statement, and
- the new value of *holiday* is received from the *Update* statement.

The example has been implemented for an Oracle 9i database using Esper (Esper, 2007) as the event processing engine. A network based SQL sensor (audit hotspot 3c) monitors the SQL statements which are sent to the database server and forwards them to the central event processing engine. The sensor is able to decode the TNS protocol used by Oracle clients to communicate with the database server. The connection information, SQL statements, bind variables and result sets are extracted by the sensor. The event engine then correlates the incoming events and displays the detected change events. The correlation algorithm must be specified using EQL (Event Query Language) in Esper. Figure 3 shows the used EQL statement.

```
   insert into SEL(emp_id, emp_name,
emp_holiday)
   select
     rsRow[0].col[0] as emp_id,
     rsRow[0].col[1] as emp_name,
     rsRow[0].col[2] as emp_holiday
   from SqlNetworkSensorEvent
   where sqlType = 'SELECT'
     and isObjectUsed('EMP')

   insert into UPD(new_holiday, emp_id,
transactionTime)
```

```
select
  bindvariable[0] as new_holiday,
  bindvariable[1] as emp_id,
  transactionTime
from SqlNetworkSensorEvent
where sqlType = 'UPDATE'
  and isObjectUsed('EMP')


select
  B.transactionTime,
  A.emp_name
  ||': Holiday changed from '
  || A.emp_holiday
  ||' to '
  ||B.new_holiday
  ||' days.'
from pattern [
  every A=SEL -> B=UPD
  (A.emp_id = B.emp_id)
  where timer:within(2 sec)
]
```

Figure 3: EQL for change history log.

As the SQL statements processed by the event engine must comply with some conventions (for example, the *Select* clause must start with the attributes *id*, *name* and *holiday* of the table *emp*), this solution is most effective if the change is performed through an application, which always uses the same fragments of SQL statements. Changes initiated via ad hoc queries, will probably not be detected with this approach. But in this case it is straight forward to generate an alert event that notifies the data auditing officer that such an action has been performed. The generated change events can be seen in Figure 4.

4 DATA AUDIT SCENARIOS USING CEP

In this section, two scenarios are presented how the introduced auditing solution can be used in business applications.

4.1 Business Process Recognition

The sensor introduced in Section 3 (audit hotspot 3c) scans the communication between the application server and the database. All SQL statements sent to the database are audited. Very often, business functions can cause a flood of SQL statements to be sent to the database server. In general, these SQL statements are meaningless to data audit officers. It is desirable to map the flood of SQL statements to a few meaningful business functions understood by all groups of users.

Using an auditing solution with CEP, business processes can be derived out of the meaningless sequence of audited SQL statements. The event engine is able to detect a pattern of SQL statements that are unique for certain business processes. Extending the event engine by new patterns of SQL statements is required to detect new business processes.

4.2 Application User Identification

In multi-tier architectures, application users authentificate them self against the application server and not against the database management system. Different client transactions use the same database user or connection. The latter is called connection pooling. As a consequence, if the communication between the application and the database server is audited, the application user is not available to the auditing solution. But unambiguous application user identification is a core requirement in many regulations.

Figure 5 shows an example architecture of such a system. Clients logon to the application server with the application user. Every time the client performs a business task, the required authorization is verified to a central security database. If the verification succeeds, the client is allowed to perform its business tasks. SQL statements forming a business task are sent to the business database using the same connection.

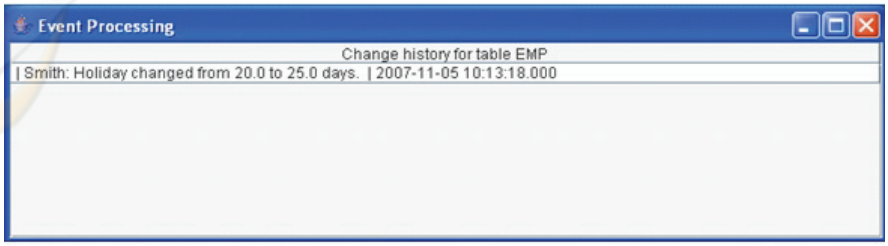

Figure 4: GUI for change event log.

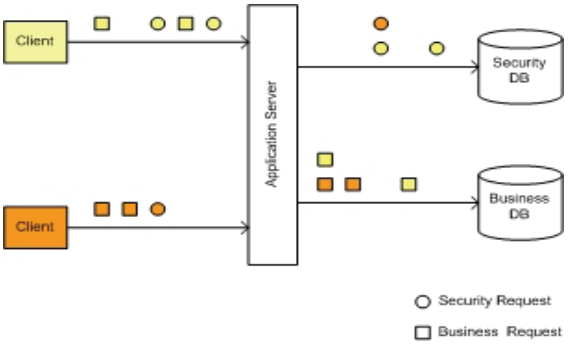

Figure 5: Connection pooling in a multi-tier environment.

To find the application user for a particular business process, security and business requests to the respective databases must be correlated. If there is currently only one application user authorized to perform a certain business task, the application user can be uniquely assigned to this task. If there are more application users authorized for the same task at the same time, at least a pool of candidate application users for this business task can be identified. Of course, this approach is only feasible if the pool of candidates is not getting too big for a certain application. On the other hand it is a non-intrusive approach to get application users in charge.

5 CONCLUSION AND FUTURE WORK

Complex event processing technology is used to support data auditing. In addition, a categorization of audit hotspots is presented. Beside custom applications, we will evaluate our approach on systems running SAP or Oracle Applications as well. Other interesting application areas beside data auditing in this context are: data firewalls, measuring quality of service and business process reengineering. Especially, we believe that the area of business process reengineering enables further opportunities for very interesting research.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge by the Austrian government, the state of Upper Austria, and the Johannes Kepler University Linz in the framework of the Kplus Competence Center Program.

REFERENCES

Agrawal, R., Bayardo, R., Faloutsos, C., Kiernan, J., Srikant, R., 2004. *Auditing Compliance with a Hippocratic Database*, in Proceedings of the 20th VLDB Conference, Toronto, Canada.

Agrawal, R., Evfimievski, A., Velu, R., 2007. *Auditing Disclosure by Relevance Ranking*, SIGMOD'07, June 12-14, 2007, Beijing, China.

Chen, S., Jeng, J., Chang, H., 2006. *Complex Event Processing using Simple Rule-based Event Correlation Engines for Business Performance Management*, in Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06), San Francisco, CA, USA.

Esper, 2007. *http://esper.codehaus.org/index.html.*

Johnson, C., Agrawal, R., 2006. *Intersections of Law and Technology in Balancing Privacy Rights with Free Information Flow*, 4th IASTED International Conference on Law and Technology, Oct. 2006, Cambridge, MA, USA.

Lee, W., Fan, W., 2001. *Mining System Audit Data: Opportunities and Challenges*, in SIGMOD Record, Vol. 30, No. 4.

Luckham, D., 2005. *The Power of Events*, Addison-Wesley, Boston.

Mangisengi, O., Pichler, M., Auer, D., Draheim, D., Rumetshofer, H., 2007. *Active warehouse: data management for business activity monitoring*, in Proceedings of the 9th International conference on Enterprise Information Systems (ICEIS 2007), Funchal, Madeira, Portugal.

Motwani, R., Nabar, S., Thomas, D., 2007. *Auditing a Batch of SQL Queries*, in Proceedings of the 21th International Conference on Data Engineering (ICDE), Istanbul, Turkey.

Natan, R., 2005. *Implementing Database Security and Auditing*, Elsevier Digital Press, Burlington, MA, USA.