Towards a Secure Data Stream Management System

Wolfgang Lindner[1] and Jörg Meier[2]

[1] MIT,
Cambridge, MA, USA
wolfgang@csail.mit.edu
[2] University Erlangen-Nuremberg
Erlangen, Germany
sijomeie@stud.uni-erlangen.de

Abstract. Todays data stream management systems (DSMSs) lack security functionality. Based on adversary scenarios we show how a DSMS architecture can be protected. We sketch a general DSMS architecture and introduce security issues that need to be considered. To face the threats we develop an extended system architecture that provides the necessary security mechanisms. We descuss the chosen concepts and illustrate how they can be realized by various system components. Our design focus is, considering the unique properties of data stream engines, to keep the impact on existing system components as little as possible and to limit the effect on the overall performance to a minimum.

1 Introduction

Data Stream Management Systems (DSMSs) have been developed over the past several years. The focus of research was on query processing and optimization [2], distribution [9] and most recently integration of data sources [3]. Security issues have not been addressed.

DSMSs differ from existing systems such as database management systems (DBMSs) in many aspects. For instance users run continuous queries which produce results by processing a continuous data stream. Without proper security mechanisms users have access to the entire system, including the ability to view and modify its behavior, data, and queries.

Because of this lack we focus in this paper on how to secure DSMSs based on the unique properties of such systems. Next, we introduce some scenarios which show the way a DSMS is used normally. In a second step we illustrate possible attacks to an unprotected DSMS.

As an example consider a DSMS that processes stock prices. It receives the changing share prices as an input data stream and executes queries of different customers based on that information. A company providing this system gets payed for delivering the results to its customers queries. We can describe the way users work with the system by the following use cases.

An administrator **sets the system up**, **connects certain data sources** containing stock price information, and **supervises the system** while it is running. The operation company might want to **integrate different data sources** from certain information providers. These sources delivering streamed data have to be attached to the system. Based on the agreement between a certain customer and the operating company the administrator **ensures that the user is able to perform the tasks** he paid for. A customer **connects to the system with a client application**, **browses through the available data sources**, **inserts queries** to the DSMS, receives the results, and uses them. A customer might want to **change a query** to get different results or to adjust the running query. He might also **insert additional queries** or **delete existing ones**. Customers might **store certain query results** in the system for later analysis, e.g., the average price of each day of certain shares.

Now imagine an adversary attacking such a system without proper security mechanisms. We consider the following scenarios:

1. The adversary connects to the system and (a) **sees all available data sources** and possibly (b) the internal state of the system **disclosing the other users' identities and their operations** including the results. The malicious user can infer from that information strategies and possible plans of competitors.
2. Not only by connecting directly to the system the adversary can read "confidential" data, but also by **intercepting the connection** of the output stream to other clients.
3. The adversary can **modify data** by inserting certain operators into the query graph. Another customer might therefore get wrong stock prices and takes decisions which may cause financial damage.
4. In addition, it is possible that the adversary (a) **fakes the incoming data** before it reaches the DSMS by changing it at any point in the network which the data passes. Also, he could (b) **pretend to be a certain information supplier** and deliver wrong source data.
5. A malicious user can perform unauthorized tasks, for example (a) **using certain operators** for data processing without having permission. That also includes (b) **administrative actions** like changing certain settings or even shutting down the whole system.
6. An attacker could **claim being a certain customer** and perform actions on behalf of that customer, e. g. deleting stored data, changing or quitting running queries. Again, that could cause financial damage to the real customer who uses the results.
7. An adversary could **fully load the system** by performing queries that consume the whole available system capacity, e. g. in terms of computational power. He might also increase his queries priorities' so that the ones of other customers can not deliver results in time.

According to [7], these adversary scenarios can be clustered in three threat categories. We associate the described scenarios to the following categories:

(C1) **Improper release of information** which can be further divided into
  (C1a) **disclosure of data** [scenarios 1a, 2] (either inside the query network or while transferring it over the network) and
  (C1b) **disclosure of system internals** [scenario 1b].
(C2) **Improper modification of data** where we distinguish between
  (C2a) **changes outside the system** [scenarios 4a, 4b] (before the input stream reaches the system or after the output stream leaves the system) and
  (C2b) **changes inside the query network** [scenarios 3, 5a, 6] (either the streaming data or the query graph).
(C3) **Denial of service attacks** [scenarios 5b, 7]

In this paper we address all of this problems. After sketching a general DSMS architecture and briefly describing common security aspects that need to be considered in Section 2, we address the illustrated security problems with proper solutions in Section 3. We show how the introduced mechanisms can be integrated in the architecture.

2 Background and Related Work

As far as we know non of the current DSMSs provides security. The following projects are examples for such data stream processing engines.

Borealis [1], an existing prototype, which is been developed at Brandeis University, Brown University, and MIT, is based on **Aurora** [2] and **Medusa** [21]. Aurora* is a distributed version of Aurora while Medusa is a federated distributed system. Many of the ideas in Borealis are developed in these two projects. **STREAM** [4] or the STanford stREam datA Manager is supposed to be a "general-purpose" DSMS and is a project of Stanford University. To express queries, a language called CQL (Continuous Query Language) is introduced. Declarative queries are compiled into a query plan. **PIPES** [13] is a project of the University of Marburg using a "hybrid multi-threaded scheduling" three layer architecture. **TelegraphCQ** [8] a general system for adaptive data flow processing with an extension to support shared continuous queries is a project developed at Berkley University.

To propose solutions for securing DSMSs we briefly sketch a general data stream architecture. Next, we discuss security issues that have to be considered.

2.1 DSMS Architecture

Our general architecture of a data stream processing system (ignoring distribution [9] and high availability [11]) is shown in Figure 1. We derived the illustrated architecture mainly from the mentioned prototypes and research projects [21, 4, 13, 8].

We differentiate between user-interaction with the system, which is shown on the top right of Figure 1, and administrative actions which take place on the top left. The later includes management tasks like connecting or disconnecting

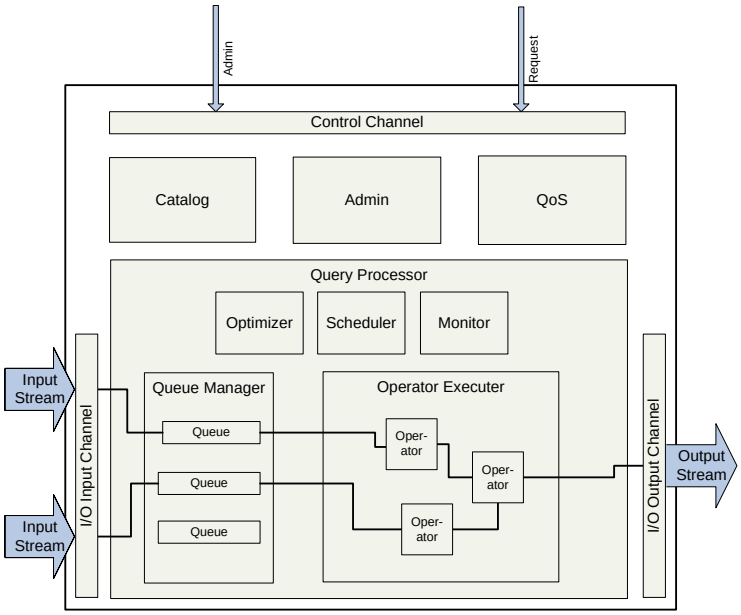

Fig. 1. Common DSMS Architecture

streams. Every request reaches the system through the Control Channel. The Query Processor (QP) is the core of the system. The actual transformation of the incoming data stream (via the I/O Input Channel) is done there by combining operator-boxes, executing them in the Operator Executer and finally, streaming the results to the I/O Output Channel. The query optimization process is controlled by the Optimizer supported by the Scheduler and the Monitor. Queues are managed by the Queue Manager. They are able to provide views on data streams as well as temporarily store data for window based operations. Queues can also be used between two operator-boxes. The Admin module controls the system, especially the QP. Every control interaction with the system is managed here. The QoS component keeps track of the overall system performance and the adherence to given QoS-requirements. The Catalog stores meta-data and query diagram descriptions. It is accessible by all components. The Catalog is consulted when a user wants to access objects in the system.

In the future the described architecture can be extended with a database attached to the DSMS. In this way persistently stored data can be processes together with streaming data and users are able to save information, e.g. calculated results (as an example see the use cases in Section 1). This extension enables the DSMS to provide the user with a service which includes traditional data management (like in DBMSs) together with stream processing capabilities. However, we do not consider this scenario due to the limited space here.

2.2 Security Issues

Deliberating to secure an information system in general involves different issues of security. According to [10, 7] this are

- Authentication,
- Authorization and Access Control,
- Confidentiality and Integrity,
- Availability,
- Auditing,
- Privacy,
- Inference security,
- Physical, hardware security, and
- Operating system security.

Next, we briefly present the concepts behind *authentication*, *authorization and access control*, as well as *confidentiality and integrity*, since they are essential for implementing any security mechanism and data protection. They address the security categories C1 and C2, which are introduced in Section 1. The remaining aspects are out of scope of this paper and not further investigated. *Availability* is related to category C3.

In accordance with [7] we use the following terms: A **subject** is a user or programs that runs on behalf of a user that accesses the system. Any entity in the system that contains data or allows operations to be executed is called an **object**. **Access controls** are responsible for ensuring that all accesses of subjects to the system objects occur according to certain security policies.

Authentication In order to distinguish between different identities (subjects), we need to *authenticate*. Authentication is any process by which the system verifies that someone is, who he claims he is. This usually involves a username and a password, but can include any other method of demonstrating identity, such as biometric attributes [7]. The unique identification of subjects is the basis of every further authorization mechanism.

Authorization Once a user is identified, the *authorization* process has to decide if the subject is permitted to access a certain resource (object) [7]. This is usually determined by finding out if that person is a part of a particular group, if that person has paid admission, has a particular level of security clearance, or has certain access rights. Authorization is based on access control, which consists of access rights and control policies.

Access Control Access Control ensures that every access to the systems occur according to certain security rules. That involves different aspects: Access Rights, Access Matrix and Control Policies and some other issues like Quality of Service and Time-based access. We illustrate them in the following paragraphs.

Access Rights In [18] a reference for database access rights is given based on the SQL99 standard: *grant*, *revoke*, *select*, *insert*, *delete*, *references*, *update*, *grant option*, *create* and *drop*. According to [15, 16] commercial database systems like Oracle [17] or Microsoft SQL-Server [14] distinguish several of these access rights. We briefly describe the most important ones:

- **Select.** The data in the system can be read. That includes the meta data (e.g. the schema) about an accessed object.
- **Insert.** Data can be added and saved persistently.
- **Delete.** Stored Data can be deleted.
- **Grant option.** Access rights can be passed to other subjects.
- **References.** The right to define foreign keys.
- **Create, Alter, Drop.** Creating, changing or deleting the schema.
- **Grant, Revoke.** The permission to change access rights.

Access Matrix and Control Policies Once a subject is authorized (or a process running on behalf of users), we want to determine whether or not the given identity is allowed to access a resource. For that reason we need to implement a relation between subjects and objects with certain access rights. As described in [19] there are two different concepts to implement an access matrix which establishes the connection between subjects and objects by storing the corresponding access rights.

- **Access Control Lists.** Each object has a list of valid subjects with their access rights. This list is called an ACL [19].
- **Capabilities.** Each subject owns a list of objects with corresponding access rights [19].

Also, [19] distinguishes between three different policies:

- **Discretionary Policies.** Discretionary protection regulates the access of subjects to objects based on identities and authorizations that specify the access mode for each subject and each object in the system.
- **Mandatory Policies.** Mandatory policies govern access based on classification of subjects and objects with security levels. A security level reflects the sensitivity of the information. The security level of a subject reflects the subject's trustworthiness. The levels are elements of a hierarchical ordered set (e.g. top secret - secret - confidential - unclassified). Depending on the security level of a subject and the one of the object the subject requests, the access is granted or denied.
- **Role-based Policies.** Role-based access controls (RBAC) regulate the access based on the activities the user executes in the system. It requires to identify roles which are associated with a set of actions and responsibilities. Access authorization on objects are specified for roles. A user playing a role is allowed to execute all accesses for that role. Roles can be hierarchically organized [19].

A temporal extension to RBAC is proposed in [6]. Temporal-RBAC adds time-based constraints to the model including periodic role enabling and disabling and temporal dependencies among such actions. A formal description and an implementation is given in [6].

Based on such a policy it is possible to store both allow-rules which give a certain right to a subject (closed system) and deny-rules which explicitly remove a right for a subject (open system) [7]. When both possibilities exist the system has to know in which order the rules should be applied.

Quality of Service Besides access rights (like whether or not a subject is allowed to perform a certain action), we also consider in which "quality" an action is executed. Think of applications where customers who want to get high quality have to pay more than others. Quality can vary in different properties. Related to stream processing, we consider the following.

- **Latency.** How fast will the answer arrive.
- **Jitter.** How big are the fluctuations in latency.
- **Bandwidth.** How much data is transferred in an given amount of time.
- **Priority.** Which priority in relation to others does the user's query has.
- **Special operators.** Which operators an user is allowed to perform, e.g. a certain user is only allowed to use an aggregation operator but no join operator.

Confidentiality and Integrity To ensure confidentiality of transferred data we have to secure communication links. These links are either inside the system among different nodes in a distributed environment or they are connections from the system to a outside point, e.g. a client. We have to make sure that only the authorized destination of a data connection is able to read the data. Another issue is data integrity which means to ensure that the information is not changed unauthorized during transfer. Both problems can be solved using cryptographic mechanisms like encryption and electronic signatures. There are existing protocols like SSL/TLS [20] and IPSEC [12] which can be used for that purpose.

2.3 Challenges in DSMSs

In contrast to discrete queries (as known from database systems) users enter **continuous queries** to process streaming data in DSMSs. As a consequence, the control- and the dataflow is separated (Control Channel and I/O Input/Output Channel).

Further, the QP arranges operators which process data streams connected to them and the optimization process **continuously adjusts the query network**. This dynamic reconfiguration has to be considered because data and operations might be merged inside the system and we have to ensure that the results a user gets suit the authorization rules.

Another aspect is the different **user abstraction level** DSMSs provide. Either there is a SQL-like interface (analog to DBMSs) or users work with the system via a box-and-arrow-semantic specifying, a data flow. The security concept has to be implemented according to the used model because the system's changed behavior reflecting the security functionality affects directly the users' interactions with the system.

An important challenge is to keep the impact of the security checks on the **overall system performance** as little as possible (system load, latency, throughput).

In Section 3 we show how to adopt existing security concepts, which are introduced in Section 2, and extend them to face the unique challenges of DSMSs.

3 Security Model

In Section 1 we illustrated how an unprotected DSMS can be attacked. We now focus on the problem of **improper release of information** (**data** or **running queries**) and **improper modification** (**outside** or **inside** the system). Based on the general DSMS architecture we propose solutions for these problem classes.

3.1 Secure DSMS Architecture

The three major mechanisms towards a secure DSMS that face the described threats are: First associating an identity to users by authentication, second deciding if and in what way access is allowed by authorization and access control, and third securing communication to ensure confidentiality and integrity.

Figure 2 shows the extended architecture including the security components. One of the design goals was to keep the impact on the existing system components as little as possible so that every module can still focus on it's specific task. The optimization process inside the QP for instance is able to work independently of the security mechanisms. Each of the introduced components, which are described in the following paragraphs, is responsible for a specific task and can be assigned to one of the described security mechanisms:

a) Associating an identity to users and ensuring that to every request for the system the corresponding subject is known
   - Session Manager
   - Authenticator
b) Deciding if and in what way access to certain objects is allowed and ensuring that a subject only gets the information it is allowed to see
   - Authorizer
   - User Abstraction Layer
   - Filter
c) Ensuring confidentiality and integrity of transferred requests and data
   - Encrypted Transport (for input and output streams)
   - Encrypted Transport (for users' and administrators' requests)

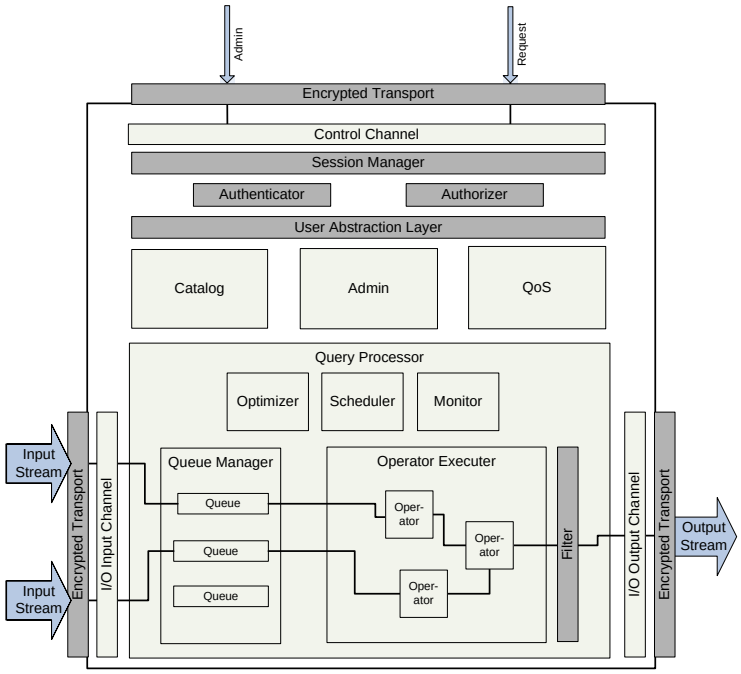

Fig. 2. Secure DSMS Architecture

Session Manager The Session Manager assigns each request to a session which belongs to a subject. This assignment is the basis for further authentication and authorization. Before the first request is accepted by the system the user has to prove his identity via the Authenticator.

Authenticator The Authenticator checks whether a user is the one he claims to be. This can be done by providing a name and password. There are different other possibilities how an user can prove his identity [7]. We do not focus on these any further. The authenticated name is mapped to an internal user-id which identifies the subject uniquely. This id is the basis for the further authorization mechanism.

Authorizer As stated before, once a user is successfully authenticated, a user-session is established including the corresponding user-id. Every further action or command the user requests has to be checked for permission by the Authorizer.

The Authorizer has to grant or deny any requested action. It implements the access control and security model illustrated in the following. This enables the system to decide whether or not a requested action on a certain object is allowed. This verification can be done before any other component is instructed to process the request.

Based on RBAC [19], we propose a security model for DSMSs that is illustrated in Figure 3. We distinguish between four entities: Users (subjects), roles, objects and permissions (access rights, e.g. read, modify, delete). Roles are associated to permissions on objects. Roles summarize certain access rights necessary to perform a certain job function. Users "can play" certain roles and they activate one or more roles in a session when they log in the system. Users get the permissions of all their activated roles.

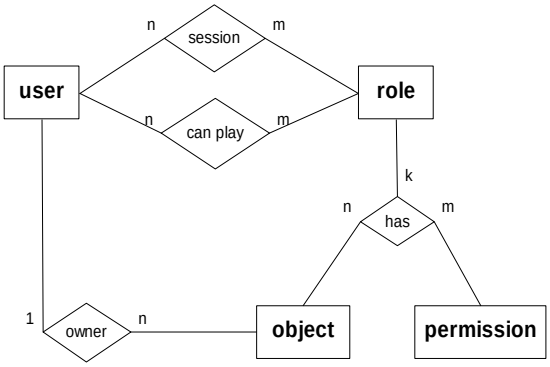

Fig. 3. OxRBAC Security Model

As in existing systems, like the Unix file systems, we add the owner relationship to the RBAC model. We refer to this model as *OxRBAC* (owner-extended RBAC). An object always has an owner which is a user. By creating an object (e.g. by inserting a query the instantiated query operators are created) the user becomes the owner of this object and gets all available access rights for it.

There are several permissions and roles which can be predefined, e.g. an administrator role which has all permissions for controlling the system, changing its behavior including altering permissions and adding other users and roles.

Furthermore we define the following rule for maximizing security: Everything which is not allowed explicitly is denied, meaning rights have to be assigned explicitly to roles by using allow-rules (closed system).

In contrast to DBMSs, were updates and insertions of data in stored relations are possible, we do not have to consider these actions in a pure data stream environment because the calculated tuples are only read by the clients. Since we believe that an extension to the general architecture will be, that DSMSs are able to store data temporally or persistently, we include the corresponding access rights in the following considerations.

We distinguish between three categories of access rights: For users, administrators, and rights relating to connected data sources:

a) User rights

- **Read.** The data on an available source stream (or on a view of it) can be read. That includes that the user sees the stream and its meta data when he is browsing through the catalog to define his query.
- **Execute.** Execution refers to operators. By having the right to a certain operator, the user can include it in his query (e.g. a join or an aggregate function).
- **Insert.** Data can be saved persistently to an attached storage.
- **Delete.** Stored data can be deleted.
- **Pass right**. Access rights can be passed to other subjects.

b) Additionally for managing the system some extended access rights must exist.

- **Attach, Detach.** Analog to tables in DBMSs (create and drop), streams must be attached to the system as data sources. The detach right allows to disconnect a stream again.
- **Create view, Alter view, Drop view.** Unlike in DBMSs where you create, alter or drop tables and views, these operations are only possible on views (on streams) in DSMSs.
- **Grant, Revoke.** To change access rights on objects, the subject (either an owner or an administrator) needs these rights on the corresponding object.
- **Shutdown, Restart, Adjust system behavior.** Administrative operations like these can be handled with execution rights on corresponding functions (which are also modeled as system objects) in the system.

c) Lastly, we manage rights related to data sources. In contrast to DBMSs where users "produce" data to store it, in a DSMS information is delivered by the streaming sources. To be able to limit the actions a stream could initiate through sending replacement or deletion tuples as described in [1], we introduce the following rights:

- **Insert.** Normally a stream inserts data to the system. Therefore the insert right is needed. Without that right a data source cannot interact with the DSMS.
- **Update.** Some sources might send correction messages for previously transferred tuples. For sending such replacement tuples the update right is needed.
- **Delete.** A special case of replacement tuples is a deletion tuple. To be able to delete a previously transmitted tuple, the delete right is necessary.

User Abstraction Layer To ensure that a subject only gets to see the objects it has permissions for, we provide individual views on the system. Such a view, which only includes objects and operations the subject is allowed to access, is provided by the User Abstraction Layer. Considering the use cases of Section 1 an user browsing the catalog or looking at running queries only sees the objects he is authorized for. The User Abstraction Layer has to establish a relation between the individual views of different subjects and the real internal

state of the system, e.g. the whole query network. This component communicates with the AUTHORIZER to check access permissions on objects.

The available interface to interact with a DSMS might be either a descriptive language (analog to SQL, like CQL [4]) or a formal description of the desired data flow, from source to destination, including the transforming operators in between (like the boxes and arrows in [2]). The USER ABSTRACTION LAYER has to provide a user-specific view on the system corresponding to the used model.

Filter The second module we introduce to avoid improper release of information is the FILTER at the end of the QP. It ensures that an output stream for a certain subject only contains data the subject is allowed to get. This is necessary because as a result of query optimization it might be possible that streams and operators of different users get combined and merged inside the QP. However, the output of the QP has to be a set of distinguished user streams. By introducing the FILTER we allow the QP and OPTIMIZER to work independently of the security checks. Not only the implementation is easier because every component provides separated services, but also the impact on performance will be smaller as we do not influence any optimization algorithm or constrain the QP in any way. Further, the access to output streams of the FILTER has to be synchronized with the corresponding request that produced the output. The request is checked for permission by the USER ABSTRACTION LAYER via the AUTHORIZER. Subsequently the FILTER has to ensure that only an allowed subject gets the corresponding query results. In that way not only the CONTROL CHANNEL is secured by authorization, but also the actual data transfer is protected by access control.

Encrypted Transport We propose to install components in the DSMS architecture to secure data transfers, both at the stream (input and output) and the request side of the system (ENCRYPTED TRANSPORT).

To ensure that data is transferred confidentially so that only the authorized participants are able to access it, we need to encrypt the data and the control channels. Referring to the secured architecture different levels of encrypting the data transfer are possible.

- **Inside the system.** The flow of information inside the system should be encrypted, especially when we assume that the query processing takes place on different nodes connected via a network.
- **Outside the system.** Both the transferred data via I/O INPUT CHANNEL and I/O OUTPUT CHANNEL and the requests for the system via CONTROL CHANNEL should be secured.

We assume that nodes of the same DSMS can be trusted and the network is under our own control. Then, the second case is the important one because the information leaves the system boundaries and we cannot be sure which way it takes to reach the client.

However, since the purpose of this paper is to develop a security framework for DSMSs, rather than showing how to adopt existing encryption algorithms, details for securing the data transfer are not considered here.

3.2 Example

Considering the illustrated use cases in Section 1, the behavior of the DSMS changes with the introduced security mechanisms in following way:

A user connects from a client to the DSMS server by using an encrypted transportation protocol. The server proves its identity by providing a certificate. After the establishment of that connection through the Encrypted Transport the user interacts with the DSMS. Every request which reaches the system through the Control Channel is associated to a session (inside the Session Manager), which is owned by that certain user. Before being able to perfom any action, the user has to log in, proving his identity. After a successful authentication process inside the Authenticator, certain roles the user is allowed to play are activated, a new session is established and every further request is connected to that session. Every action requested by the user can be checked for permission by the Authorizer now. The user browses the catalog, where he only sees objects he is allowed to access through the User Abstraction Layer. He inserts a query by using available data sources. The system integrates the inserted query in the internal query network and calculates the results. The user connects to the produced output stream, which is available at the Filter, by using another encrypted communication link to the Encrypted Transport at the output side of the DSMS. As the user gets his individual view on the system, he can look at his running queries and modify them. Finally the user logs out and closes the connections to the system.

3.3 Future Issues

For the purposes of completeness we briefly want to mention two aspects we do not further investigate because of limited space. These are secure distribution and quality of service.

Distribution Many existing DSMSs, like Borealis [1], include distribution functionality for load balancing and high availability. A distributed security concept has to provide solutions for:

- **Trusted authentication.** We have to ensure that an user logged in at one site can use the complete distributed system as one service. Either there is a single point of entry for an user to connect to the whole system, or users can interact with the distributed nodes at different sites.
- **Permission management.** The rules necessary for checking permissions have to be replicated among the participating sites so that access control is possible wherever resources are used.

- **Different administrative domains.** In federated DSMSs, like Medusa [5], sites can be under the control of different administrative domains. Data that is distributed has to be protected of unauthorized access from users working at the other site. The access control mechanisms have to work in a global way, prohibiting unwanted disclosure of information. We propose further to introduce the possibility for the users to decide whether or not data processing could be pushed to other sites.
- **Secure communication.** In a distributed environment the secure communication inside the system, as mentioned before, becomes important to ensure that only allowed processes connect to output streams of other nodes. The network connecting the sites might be untrusted.

QoS To provide a QoS-based security service like described in 2.2 the QoS module of the system has to be extended too. Depending on rules defining QoS-properties for subjects actions must be taken to guarantee them. For instance a subject might be allowed to set the priority of his query to a certain level while others are not allowed to do so.

4 Conclusion

As the adversary scenarios show there are different threats to unprotected DSMSs. In this paper we proposed solutions towards a secure system. We gave a general overview and showed which problems have to be investigated. We focused on two main threat categories: Improper release and improper modification of data. We described the concepts which are necessary to solve these problems. Based on the introduced system architecture we illustrated how the security mechanisms can be implemented. The third problem, denial of service attacks, was partly solved by the authorization process as we did not consider auditing and QoS-related security features in our solution.

The concept we propose in this paper is generic enough to be integrated into any existing data stream management system and therefore an ideal basis for enterprise architectures.

Currently, we are building a first prototype by implementing the proposed security features into Borealis [1], proving that a DSMS can be secured without creating too much of a performance overhead.

References

1. D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.
2. D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 2003.

3. D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *VLDB*, 2004.
4. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin*, 26(1), 2003.
5. M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *NSDI*, 2004.
6. E. Bertino, P. A. Bonatti, and E. Ferrari. Trbac: A temporal role-based access control model. *ACM TOISS*, 4(3), 2001.
7. S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison Wesley, 1994.
8. S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.
9. M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.
10. D. K. Hsiao, D. S. Kerr, and S. E. Madnick. Privacy and security of data communications and data bases. In *VLDB*, 1978.
11. J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik. High-Availability Algorithms for Distributed Stream Processing. In *ICDE*, 2005.
12. IETF. IPSec. http://www.ietf.org/html.charters/ipsec-charter.html.
13. J. Krämer and B. Seeger. Pipes - a public infrastructure for processing and exploring streams. In *SIGMOD*, 2004.
14. Microsoft. Microsoft sql server. http://www.microsoft.com/sql.
15. Microsoft. Sql server 2000 sp3 security features and best practices: Sql server 2000 security model. http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sp3sec01.mspx.
16. P. Needham and S. Iyer. Oracle database 10g security and identity management, 2003. http://www.oracle.com/technology/deploy/security/pdf/twp_security_db_securityoverview_10r1_1203.pdf.
17. Oracle. Oracle database. http://www.oracle.com/database/index.html.
18. R. Ramakrishnan and J. Gehrke. *Database Management Systems*, chapter Security and Authorization. Mc Graw Hill, 3rd edition, 2003.
19. Ravi S. Sandhu and Pierrangela Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9), 1994.
20. RFC. Rfc2246 tls. http://www.ietf.org/rfc/rfc2246.txt.
21. S. Zdonik, M. Stonebraker, M. Cherniack, U. Çetintemel, M. Balazinska, and H. Balakrishnan. The Aurora and Medusa Projects. *IEEE Data Engineering Bulletin*, 26(1), 2003.