Mandatory and Location-Aware Access Control for Relational Databases

Michael Decker

Institute AIFB, University of Karlsruhe (TH)
Kaiserstr. 89, 76 128 Karlsruhe, Germany
decker@aifb.uni-karlsruhe.de

Abstract. Access control is concerned with determining which operations a particular user is allowed to perform on a particular electronic resource. For example, an access control decision could say that user *Alice* is allowed to perform the operation *read* (but not *write*) on the resource *research report*. With conventional access control this decision is based on the user's identity whereas the basic idea of Location-Aware Access Control (LAAC) is to evaluate also a user's current location when making the decision if a particular request should be granted or denied. LAAC is an interesting approach for mobile information systems because these systems are exposed to specific security threads like the loss of a device. Some data models for LAAC can be found in literature, but almost all of them are based on RBAC and none of them is designed especially for Database Management Systems (DBMS). In this paper we therefore propose a LAAC-approach for DMBS and describe a prototypical implementation of that approach that is based on database triggers.

Keywords: Location-based Services, Database Management Systems (DBMS), Mandatory Access Control (MAC), Mobile Computing, Security Models.

1 Introduction

Location-based Services (LBS) [14] are services for mobile computers (e.g. PDA, smartphones) that evaluate the user's location to provide the service. A standard example for a LBS is that of a Point-of-Interest-Finder service which guides a user to a facility of a particular category (e.g. restaurant, monument, pharmacy) in his nearer surrounding. The realization of LBS is enabled by the availability of techniques to determine a mobile computer's location. Examples for such locating systems are the satellite-based *Global Positioning System (GPS)* or special systems for indoor locating based on infrared light or ultrasound waves. A good overview concerning various locating technologies can be found in [14]. Our work concentrates on a special kind of LBS called *Location-Aware Access Control* (LAAC). Access control in the domain of computer systems means to determine if a user is allowed to perform a particular operation on a particular resource or not [11]. For example, the access control component of a computer

R. Mehmood et al. (Eds.): EuropeComm 2009, LNICST 16, pp. 217–228, 2009.

system could prohibit that user Alice performs the operation *read* on the resource *payroll file*. The special idea of LAAC is that for an access control decision also (or even only) the current location of the user is considered [8]. Using LAAC we could enforce the policy that a user is only allowed to access a file with confidential business data while he stays at the premises of the company.

To implement access control a special data model is needed to manage the rules in a way that can be understood by human administrators as well as the computer system; such a model is called *Access Control Model* (ACM). A simple form of an ACM is the *Access Control Matrix* where each row represents one user and each column one resource [11]. Each element in the matrix then lists operations the respective user is allowed to perform on that resource.

In this article we introduce a novel ACM for Database Management Systems (DBMS) that is location-aware and follows the principle of Mandatory Access Control (MAC). In MAC-based systems users and data objects have security labels. The system then allows or forbids operations a user wants to perform on a particular resource depending on rules that evaluate the label of both resource and user. While there are several papers proposing location-aware ACM (LAACM) that are extensions of RBAC, there are almost no publications dealing with LAACM that implement the MAC or DAC concept. Further there is considerable work concerning ACM for DBMS (e.g. [10,16]), but none of them is location-aware.

The remainder of the article is organized as follows: In the next section 2 we discuss some basics concerning access control and relational database management systems. For the description of a LAACM we need a location model that is introduced in section 3. The main contribution of our article is a location-ware ACM for DBMS in section 4. Section 5 is devoted to a description of a prototypical implementation of the model. Related work is covered in section 6 before we conclude in section 7.

2 Basics

2.1 Mandatory Access Control

In the pertinent literature like [11] usually three groups of ACM are distinguished: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access Control (RBAC). Since the novel ACM proposed in this paper is a MAC model we give only a superficial discussion of DAC and RBAC before MAC is introduced in more detail.

DAC is the approach implemented by most conventional file systems (e.g. those of MS Windows and Linux): The creator/owner of a resource has all permissions for a resource and it is on his *discretion* to give permissions on this resource to other users (and maybe revoke them later). For example, if *Alice* created a file she could grant the permission for the operation *read* to her colleague *Bob*. The *Access Control Matrix* mentioned above is an ACM for DAC.

The basic idea of RBAC is that job descriptions within an organization rarely change but the assignment of jobs to people changes quite often, e.g. employee

hired, fired or promoted [11]. So there are *roles* as mediators between users and permissions: roles are created according to job descriptions (e.g. role "secretary", role "manager") and are assigned to the set of permissions that is necessary to fulfil that job. Then users are assigned to roles; however, it is forbidden to assign users directly to permissions. Contemporary variants of RBAC also support *Role Hierarchies* and *Separation of Duties (SoD)*, i.e. sets of roles can be defined that are mutually exclusive at administration time (static SoD) or at runtime (dynamic SoD).

The "mandatory" in "MAC" means that access control rules are imposed by the system without the demand that end users have to (or even can) configure permissions on resources. To implement MAC we need security labels that are assigned to users and resources. If a user wants to perform an operation on a resource the MAC-system will make its decision based on a consideration of both the user's and the resource's labels. To exemplify this we assume the following ordered set of security labels: $Top\ Secret \succ Secret \succ Confidential \succ Public$. If a resource (say an electronic document) is classified as *Secret* then a user with clearance *Secret* or even *Top Secret* would be allowed to read that document; however, a user with lower clearance like *Confidential* or just *Public* wouldn't get read access.

This read rule is called *no-read-up* and actually enforced by the most prominent MAC model, namely the Bell-LaPadula-model [11]. The model has also a rule for write operations which seems to be counterintuitive at the first glance: a user is only allowed to write to a resource if the resource's security label is higher than the user's security clearance. The reason behind this *no-write-down* rule is that it should be prevented that a user (or a Trojan Horse working with the user's permissions) reads data and writes them to resources with lower security level so that users with a lower clearance than that of the original data could gain access. The security labels for the Bell-LaPadula model have not only the above mentioned hierarchical component, but also a non-hierarchical component. This non-hierarchical component is an unordered set of thematic categories like *finance*, *chemistry* or *nuclear*. If we have two security labels $X = (A, B)$ and $Y = (A', B')$ where A and A' are elements of the hierarchical component and B and B' is from the non-hierarchical component then X dominates Y iff: $Y \preceq X \quad \Leftrightarrow \quad (A' \preceq A \wedge B' \subseteq B)$.

While the Bell-LaPadula-Model is a MAC-model to ensure the secrecy of data there are also models that were developed for the purpose of enforcing the integrity of the managed data. The most prominent model for integrity is the Biba-model [2]: one of its rule is called *no-write-up*, i.e. a user is not allowed to write into an object with a higher integrity level. The rationale for this is that it should be prevented that a user at a low integrity level "pollutes" a data object at a higher integrity level with inferior data.

MAC is usually combined with DAC to absorb configuration mistakes in the DAC configuration. For many years MAC was only applied to secure information systems used by organizations like military and intelligence service. However, nowadays there are MAC implementation for "ordinary" computer systems available, e.g. Security-Enhanced Linux (SE-Linux). In section 6 presenting "Related Work" we will also mention a few database systems that support MAC.

2.2 Basics of DBMS

A DBMS is a special software system designed to store and manage a large amount of structured data records in an efficient way [10]. The management aspect includes especially the capability for the retrieval of data records that meet dynamically specified conditions. Further, a modern DBMS should be able to handle concurrent requests by many users. Nowadays the prevalent paradigm for DBMS is the relational model: the basic idea of this concept is that data is stored in tables; a database can be considered as a collection of tables (also called *relations*). Each table defines columns of a specified data type and stores the data records as rows. It is further possible that one record of a table references a row in another table (foreign key relationship).

More formally, the tables and their actual rows are considered as relations of a particular relation schema $R(A_1, A_2, \ldots, A_n)$. Each attribute A_i is the name of one column that contains values of a particular domain denoted by $dom(A_i)$. Such a domain could be the set of all unsigned 32-bit-integer numbers or of all character strings with a maximum length of nine characters. For a relation schema R the function $r()$ returns the relation or the current state of a table, i.e. all the tuples or rows t_i: $r(R) = \{t_1, t_2, \ldots, t_m\}$. To refer to the value of attribute A_j of tuple t_i we write $t_i.A_j$. The set of all relation schemas is denoted by $\mathbb{R}$, i.e. $\mathbb{R} = \{R_1, R_2, \ldots, R_k\}$.

Today most DBMS support the Structured Query Language (SQL), a declarative command-language to manage the database's structure (e.g. to set up new tables), to insert, update and query the data and for administrative commands like granting permissions on database objects to users. DBMS usually have also some kind of access control that follows the DAC approach: The creator of a database object (e.g. a table) as owner of that object can grant permissions on that object to other users. For a table for example one usually can grant permissions for individual operations like viewing (selecting), updating, inserting or deleting data rows. This is supported by special SQL commands, namely GRANT and REVOKE.

3 Location-Model

For our work so far the following simple location model is sufficient (figure 1): The geometric locations of the model describe a polygon that is a non-empty subset of the reference space called *universe*. Since with a polygon each spatial extent can be approximated as precise as demanded the restriction to use only polygons is a weak one. A polygon together with a name like *London* or *England* is called *location instance* or *location* for short. Locations are grouped in *location classes*: each location instance belongs to exactly one location class. This principle can be also found in the geographic markup language (GML) where each feature (object with spatial dimension) belongs to a feature type [15]. Also many systems that provide support to visually work with geographical data like maps (e.g. Geographic Information Systems (GIS)) support the concept of several layers that can be switched on and off individually; each of these layers usually contains

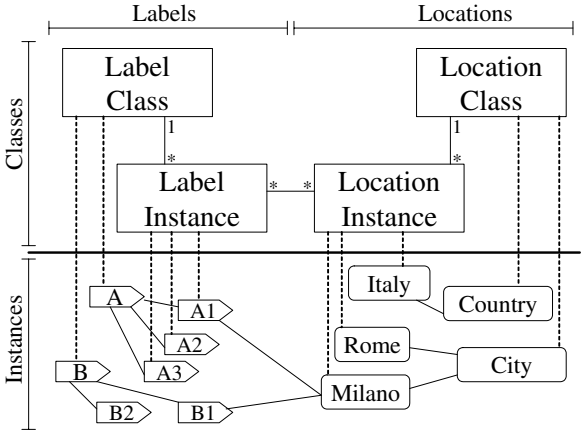

Fig. 1. Location model

the spatial objects of one type (or location class), e.g. there is one layer for the streets and another one for forest areas. From the perspective of object-oriented software design this is also a natural principle.

Formally, L is the set of all location classes and l the set of all location instances. The function $\lambda()$ maps a location class to the set of all its instances. With $\mathbb{P}(l)$ as the power set of all locations instances (i.e. the set of a all possible subsets of l including the empty set $\emptyset$) this can be written as follows: $\lambda : L \rightarrow \mathbb{P}(l)$.

To exemplify the model we consider the following set of location classes: $L = \{universeclass,\ cities,\ countries\}$ For location class *cities* we have two instances $\lambda(cities) = \{London,\ Milano\}$ and for location class *countries* there are also two instances: $\lambda(countries) = \{U.K.,\ Italy\}$. For all the location instances of a given location class two properties have to hold: (1) Two location instances do not overlap spatially; (2) The union of all location instances covers the universe completely. These two properties guarantee that for a given location class and a given point we always can determine exactly one location instance belonging to that class which covers this point.

We further need different classes of security labels which in turn have instances. A security label class corresponds approximately to the compartments (component with unordered labels) found in the Bell-LaPadula-model [1]. Using the security label instances of one security class we can classify documents and locations under the view of a particular topic or domain. The number of different labels for each class may vary and is defined in the security class whereas the strictest label has always the value "1"; the remaining classes have the following natural number $2, 3, \ldots, n \in \mathbb{N}$. As example we consider a company operating plants in all European countries that produces two types of main products A and B. In some countries the company has to fear espionage because there are local companies that are also active in the market for these product. The company therefore defines two classes of labels: $S = \{A, B\}$. To obtain the subset

of label instances from the set of all labels s for a given class from S there is function: $\sigma : S \rightarrow \mathbb{P}(s) \setminus \emptyset$. Label class A is used to classify data records and location instances according to product A and has $|\sigma(A)| = 3$ labels, namely $\sigma(A) = \{A1, A2, A3\}$. Countries were no espionage has to be feared will get a clearance of $A1$; if a mild level of espionage has to be feared because of a small competitor in a country then the country will get the label $A2$; however, if strong espionage activities have to be assumed the label $A3$ will be assigned as clearance to the country.

4 LAACM for DBMS

The novel ACM for DBMS proposed in the article at hand supplements the basic DAC by allowing to assign location-constraints (LC) to individual rows in a table. A row with such a location-constraint can only be accessed when the user currently stays at a location which is in accordance with that LC. Because we are following the MAC approach these LC are created automatically by the runtime system when the respective data row is created and inserted into a table. Further it is not possible to alter or remove such a location-constraint by an usual administrative operation. Our MAC models supports *direct* and *indirect* LC so far:

Direct LC (DLC): For this case each individual row of a table obtains a LC that restricts the access to that row to users who currently stay within a particular location-instance. To activate the creation of DLC for a table $R \in \mathbb{R}$, the table has to point to one location class: $\rho_1 : \mathbb{R} \rightarrow L \cup \emptyset$. When a user inserts a new row into this table the instance $l \in \lambda(L)$ that covers the current location of that user is looked up and assigned to this row as location constraint.

Indirect LC (ILC): For this case security labels can be attached to each individual row of a table. To activate the creation of ILC the table R has to point to at least one security label class: $\rho_2 : \mathbb{R} \rightarrow \mathbb{P}(S)$. When a user inserts a new row to a table the runtime system determines all location instances that cover the user's current position. Since it is demanded that two location instances of the same location class don't overlap each other, this implies that we'll get not more than one instance of each location class. Afterwards it is evaluated if security labels belonging to one of the security classes returned by $\rho_2(R)$ are assigned to the location instances. If such labels are found they will be attachted to the newly inserted row. Access to this row will then only be granted when the user stays in a location whose security labels are at least as high as those assigned to the row.

To obtain a list of of all LC assigned to a given row of a table we introduce the functions τ_1 and τ_2. The first function is for DLC and returns DLC the locations where access to that row is allowed: $\tau_1 : t \rightarrow \mathbb{P}(l)$. If no DLC is assigned to that row the empty set will be returned. This means that the respective row can be accessed at all locations. The second function τ_2 returns a possible empty set of all security labels assigned to a row: $\tau_2 : t \rightarrow \mathbb{P}(s)$.

5 Implementation

For the prototypical implementation we chose PostgreSQL[1] (PG) for several reasons: PG is known as the most powerful open source DBMS and provides a good implementation of the pertinent standards. Its PostGIS-plugin[2] offers an excellent implementation of the OGC's *Simple Features Specification* [15], which is an extension for SQL to work with spatial data.

We developed an implementation of the proposed ACM using PG's support for procedural programming, namely the PL/pgSQL language to write so called *Stored Procedures*. Stored procedures not only encapsulate sequences of conventional SQL commands, they also provide the common features of modern high-level programming languages like loops, variables and if-then-else-decisions. The implementation and the source code fragments presented in this section will only work with PG, but it should be easy to transfer the basic principle to most other modern DBMS.

Our implementation (figure 2) relies heavily on database triggers: triggers allow the definition of stored procedures that are invoked when particular events occur. For our work the events of interest are insertion-, update- und delete-operations on tables. In the body of a trigger procedure there are variables to query the name of the table that raised the event or to access old or new data rows. Using trigger procedures we can prevent the execution of an operation on a database table. This is of interest for update and delete operations on a table row that should be inhibited according to our MAC because the mobile user doesn't stay at the right location. A trigger for insert operations can be employed to set up the location restriction upon creation of a new row in a location-aware table. To prevent read-operations of table rows we would need to set up a triggers on select. However, neither the SQL standard nor PG support select-triggers. As workaround for this problem we resort to *views*. A view presents the result of a SQL-statement like a real table. In this way subsets or combinations of data rows stored in tables can be offered to the user. The command to create a view named `v_table1` which hides the rows from the user according to our model is:

```
CREATE OR REPLACE VIEW v_table1 AS
  SELECT * FROM table1 WHERE is_access_allowed(oid,tableoid);
```

This view returns all the columns of *table1*, but may hide rows when the function `is_access_allowed()` returns *false*. The system parameters `oid` and `tableoid` are passed to this function: `oid` is a row number that is assigned for each row in a table. However, it is not guaranteed that `oid` is unique over all tables. That's why we also consider `tableoid`, which is an unique id for each table so a pair of this two id-numbers uniquely identifies a table row in a PG database. We use `tableoid` as column in a system table named `table2locclass` that implements the mapping ρ_1 that can assign a location class to a table to activate DLC. An DLC for a table row is stored as row in table named `DLC` with a pair (`oid`,`tableoid`) and a reference to a location instance.

[1] `http://www.postgresql.org`
[2] `http://postgis.refractions.net`

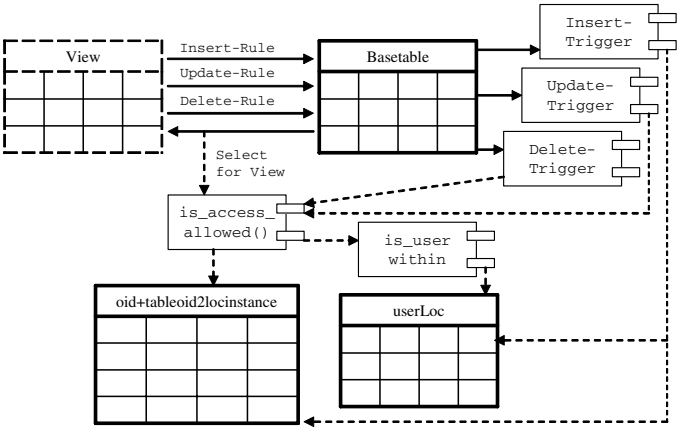

Fig. 2. Main components of the implementation

However, the user should be able not only to query data from a view but also to alter the data of a view respective the basetable behind the view. To enable this PG's rules-feature can be used to redirect insert, update and delete commands performed on the view to the respective basetable. For example, the rule for the insert command looks like this:

```
CREATE OR REPLACE RULE "ins2tab1rule"
    AS ON INSERT TO table1 DO INSTEAD
    INSERT INTO table1(id,name) values(NEW.ID,NEW.name)
```

To work with spatial data we employ PG's PostGIS-plugin which provides an implementation of the OGC's SQL extension for spatial data. As example we take the table `locinstance` which is used to store the location instances (set l according to our formalization): We first create the table using the SQL's conventional `CREATE`-command and then add a column named `polygon` for storing the location instances with the following function call:

```
SELECT AddGeometryColumn(
  'locinstance', 'polygon', -1,'POLYGON', 2);
```

The parameters have the following meanings (in that order): (1) name of the target table, (2) name of the column to create, (3) numeric code (SRID) specifying the employed coordinate system, (4) data type of the new column and (5) dimension of the employed data type. To insert a row into this table we use the following statement:

```
INSERT INTO locinstance values (1, 'District1', 4,
    GeometryFromText('POLYGON((0 0, 5 0, 5 10, 0 10, 0 0))'));
```

The elements in the values-expression specify the following things: (1) ID of the location instance, (2) name of the location instance, (3) integer specifying the location class (foreign key reference to table `locclass`), (4) definition of the polygon using a string expression following the *Well-Known Text (WKT)* representation by

enumerating the points that span the polygon. The user's current position is stored in a table called `userloc` where each tuple contains amongst other attributes a user name and a point. We assume that the employed locating system (e.g. GPS) updates this value when appropriate. With the `&&`-operator we can test if two geometries overlap. As example we show the statement that lists for each combination of location class and user the location instance that covers the user's current location.

```
select * from userloc, locclass, locinstance
    where locinstance.locclass=locclass.id and
    locinstance.polygon && userloc.location
```

Another approach for the implementation of the model would be to use PGACE (PostgreSQL Access Control Extension), an interface offered by PG to implement custom access control. This interface was introduced for the implementation of SE-PG, the MAC extension of PG (see next section). However, using this interface requires programming in C/C++ which is more prone to errors and doesn't allow the comfortable usage of PostGIS. Also PGACE is a PG-specific interface so that a MAC system implemented with PGACE cannot be transferred to other DBMS implementations.

6 Related Work

Gallagher [12] developed an extension for SQL's DAC that allows to specify location constraints when granting and revoking permissions to users. Using this extension a grant-command can have an *inside*-clause to restrict the permission assigned to a user to a particular region, e.g. GRANT SELECT ON TAB1 TO ALICE INSIDE AREA1. The meaning of this command is to allow user alice to perform the select-command on table `tab1` when she is inside the location referred by `area1`. To the best of our knowledge Gallagher's work is the only approach for LAAC especially for DBMS so far.

In a paper by Casati et al. [3] a DBMS with triggers is employed to implement autorization constraints in workflows by using triggers to enforce workflow-specific access rules like "separation of duties": if a user performed a step of a workflow instance he is not allowed to perform a particular other step for the same workflow instance, e.g. for the handling of the order the company's policy could demand that the user who performed "prepare shipment" mustn't perform "check shipment" for the same workflow instance because a shipment should always be seen by two different employees before it is dispatched. So if a user performs a task that implies a "separation of duties" rule a trigger inserts a row into a table that records which users for which workflow instances are not allowed to perform particular activities.

In [17] the only proposal of a LAACM following the MAC-approach which we are aware of can be found; however, this model wasn't designed with DBMS as special focus. In this model, security levels are assigned to users, electronic resources and locations. If a location lies within another one the inner location needs to have at least the security level of the outer location, e.g. if a building is classified as "Secret" an room within that building cannot be classified below,

e.g. as "Confidential". Further, the model assumes that resources are classified manually and prohibits that a resource is stored at a location that has a lower security level than the resource itself.

The most prominent example for a DBMS with MAC is *Seaview* [16]. This model considers secrecy and integrity, however, for the sake of simplicity we only discuss secrecy here. To each row and each single attribute instance a security label is assigned. Depending on his security label a user may not be able to see all rows in a table or all attributes in a row. However, if he wants to insert a row into a table that has the same primary key as a row invisible to him this operation cannot be rejected because in this case the user could infer that there is a hidden row. This means that the row has to be inserted regardless of the fact that this leads to multiple rows with the same primary key (but with different security labels). The term for this phenomenon is *polyinstantiation*. Polyinstantiation can also occur when rows are updated. Further, it is demanded that if a user is able to see any of the elements of a row he is also able to see the primary key elements of that row, i.e. every non-key attribute in a row has to dominate the primary key attributes. Some other MAC models for DBMS can be found in [4].

There are some research contributions that propose location-aware extensions of ACM. Almost all of these works cover RBAC-extensions, e.g. GEO-RBAC, SRBAC, LoT-RBAC or STRBAC. GEO-RBAC [6] allows to switch roles on and off depending on the mobile user's current location. However, in SRBAC the assignment between individual roles and permissions is enabled/disabled according to the user's current location [13]. A survey providing on these models can be found in [8]; this paper also discusses some scenarios for LAAC. However, none of these models provides special support for DBMS.

SE-PostgreSQL is a variant of PG that provides MAC but it runs only on SE-Linux. The strict coupling between DBMS and OS enables to keep the security context of users and data objects between the OS and the DBMS. For example, if a user reads a classified file from the filesystem of SE-Linux and stores it into a database table the classification of this file isn't lost. But there is also another lightweight MAC-implementation for PG called *Veil*[3] that doesn't require that the PG installation runs on SE-Linux. SE PG doesn't provide location-aware MAC. For the other popular Open-Source DBMS, namely MySQL, no MAC-support was implemented at all. MAC is available for the two most popular commercial DBMS, namely *IBM DB2* and *Oracle*, but these MAC implementations do not support location-awareness, too. MAC was introduced as *Label Based Access Control* in Version 9 of *IBM DB2*. The other popular commercial DBMS, Oracle, also offers MAC and calls it *Fine Grained Access Control (FGAC)*. These MAC implementations offer several types of security labels, e.g. as unordered or ordered lists and as hierarchical components. But neither DB2 nor Oracle support location-aware MAC.

Implementing access control based on the determination of a user's location raises the question how resistant a locating system is with regard to manipulation attempts. Deliberate manipulation attempts to forge a mobile user's location are

[3] `http://veil.projects.postgresql.org`

called *location spoofing*. One method to prevent this kind of spoofing is to demand that the mobile device sends some kind of information to the backend system that can only be received at the alleged location, e.g. signals emitted by beacons, e.g. [5]. Another method is to measure the time needed by a mobile device to send back a response that is based on a request that cannot be predicted, e.g. [18]. These *request-response-protocols* are especially interesting if radio waves are employed, because currently no technique is know to send data at a higher speed. An overview on different approaches to harden locating systems against spoofing attacks can be found in [9].

If a mobile user is constantly located for the purpose of LAAC by his employer's information system this raises concerns regarding data protection. In [7] we therefore survey several approaches to tackle this *location privacy* problem.

7 Conclusion: Summary and Outlook

We motivated the need location-aware access control and introduced a novel Access Control Model (ACM) for database management systems. The ACM is a mandatory access control model, i.e. its works "behind the scenes" and doesn't require manual configuration by the user. It allows to restrict the access to individual rows stored in database tables to particular locations.

Ideas for further work include rules distinguishing different operations or assigning different location restrictions to different user groups, e.g. a service technician is only allowed to edit the document in the building where it was created while an executive manager can do this as long as he is in the same country. Our model so far only supports positive restrictions, i.e. it makes a statement where something is allowed. But it is also thinkable to have additional negative restrictions, i.e. restrictions that state where something is forbidden. However, when positive and negative restrictions can be used together contradictions may occur that have to be handled.

References

1. Bell, D.E., LaPadula, L.J.: Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, The MITRE Corporation (1976)
2. Biba, K.J.: Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, The MITRE Corporation (1976)
3. Casati, F., Castano, S., Fugini, M.G.: Managing Workflow Authorization Constraints through Active Database Technology. Information Systems Frontiers 3(3), 319–338 (2001)
4. Castano, S., Fugini, M., Martella, G., Samarati, P.: Database Security. Addison-Wesley, Wokingham (1994)
5. Cho, Y., Bao, L., Goodrich, M.T.: LAAC: A Location-Aware Access Control Protocol. In: Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, pp. 1–7 (2006)
6. Damiani, M.L., Bertino, E., Perlasca, P.: Data Security in Location-Aware Applications: An Approach Based on RBAC. International Journal of Information and Computer Security 1(1/2), 5–38 (2007)

7. Decker, M.: Location Privacy – An Overview. In: Proceedings of the International Conference on Mobile Business (ICMB 2008), Barcelona, Spain. IEEE, Los Alamitos (2008)
8. Decker, M.: Location-Aware Access Control: An Overview. In: Proceedings of the Conference on Wireless Applications and Computing (WAC 2009), Carvoeiro, Portugal, pp. 75–82 (2009)
9. Decker, M.: Prevention of Location-Spoofing. A Survey on Different methods to Prevent the Manipulation of Locating-Technologies. In: Proceedings of the International Conference on e-Business (ICE-B), Milano, Italy, pp. 109–114. INSTICC (2009)
10. Elmasri, R., Navathe, S.: Fundamentals of Database Systems, 4th edn. Pearson, Boston (2004)
11. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: Role-Based Access Control, 2nd edn. Artech House, Boston (2007)
12. Gallagher, M.: Location-based authorization. Master's thesis, University of Minnesota (2002)
13. Hansen, F., Oleshchuk, V.: SRBAC: A Spatial Role-Based Access Control Model for Mobile Systems. In: Proceedings of the Nordic Workshop on Secure IT Systems (NORDSEC), Gjovik, Norway, pp. 129–141 (2003)
14. Küpper, A.: Location-based Services – Fundamentals and Operation. John Wiley & Sons, Chichester (Reprint, 2007)
15. Lake, R., Burggraf, D.S., Trninic, M., Rae, L.: GML. Geography Mark-Up Language. Foundation for the Geo-Web. John Wiley & Sons, Chichester (2004)
16. Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M., Shockley, W.R.: The seaview security model. IEEE Trans. Softw. Eng. 16(6), 593–607 (1990)
17. Ray, I., Kumar, M.: Towards a Location-based Mandatory Access Control Model. Computers & Security 25(1), 36–44 (2006)
18. Sastry, N., Shankar, U., Wagner, D.: Secure Verification of Location Claims. In: Proceedings of the 2nd ACM Workshop on Wireless Security (WiSE 2003), San Diego, California, USA, pp. 1–10 (2003)