A PRACTICAL TASK-BASED APPROACH TO ACCESS CONTROL CONFIGURATIONS

Rukshan I. Athauda and Euijoon Ahn

Working Paper Series, Number 4, November 2013

ABOUT THIS SERIES

The Applied Informatics Research (AIR) Group is a cross-disciplinary, multi-institutional collaboration based in the School of Design, Communication & Information Technology at the University of Newcastle, Australia. The principal aim of the AIR Group's Working Paper Series is to disseminate the research and/or technical output of the group in an easily accessible format. The content of Working Papers generally falls into one of the following categories:

- *Preliminary findings or results, the release of which is intended to stimulate debate and/or discussion to assist in the further development of the research*
- *Technical reports associated with applied research that may be written in a less academic style than usually published in academic journals*
- *Extended versions of published works, often containing additional implementation/ application detail, figures and tables.*

The opinions or conclusions expressed in the Working Paper Series are those of the authors and do not necessarily reflect the views of the AIR Group as a whole.

A PRACTICAL TASK-BASED APPROACH TO ACCESS CONTROL CONFIGURATIONS

Rukshan I. Athauda
School of Design, Communication, and Information Technology
The University of Newcastle
Callaghan 2308, NSW, Australia

Rukshan.Athauda@newcastle.edu.au

Euijoon Ahn
ECNESOFT PTY LTD
Unit 14, 20-30 Stubbs St,
Silverwater, NSW 2128

Osmond.ahn@ecnesoft.com.au

ABSTRACT

Configuring optimal access control is a difficult task in today's complex IT environments. Too restrictive access control leads to frustration by users, while excessive privileges leads to vulnerabilities. Unfortunately, the problem of verifying safety – i.e. no rights can be leaked to an unauthorised principal - for an arbitrary configuration of a general access model is shown to be undecidable. In this paper, a practical methodology and framework is proposed to elicit access control rights stealthily while users perform tasks in a test environment that mimic a real-production environment. To illustrate the feasibility of the framework, a prototype is implemented and presented.

1 Introduction

A typical enterprise today contains a multitude of heterogeneous IT systems that span across different networks. The complexity of IT infrastructure makes configuring and maintaining such systems increasingly challenging. This is especially true in terms of security configurations that need to be tuned to optimize protection and to block prospective attacks, while also balancing security, flexibility, and performance. This is "extremely burdensome" not only for regular users but also for experienced administrators, who have to be "very lucky" to get things working right all the time [8].

A key task for IT administrators in configuring IT systems is providing access to resources required by various users and groups (i.e. access control configurations). With a variety of systems, different user groups and access rights configurations, this task is neither trivial nor straight-forward. Misconfigurations with too excessive or too restrictive access control configurations are common. A direct consequence of excessive access control configurations is access to unauthorised resources. Unauthorised access to information and theft of proprietary information amounts to the largest dollar amount of losses by type [3].

What is an optimal access control configuration? An optimal access control configuration should allow access to *only* the required resources for *authorised* users. That is, in optimal access control configurations, (1.) access is *not too restrictive* that it disallows access to required resources for authorised users and (2.) it is *not too excessive* that it allows access to unauthorised resources.

Overly restrictive access control configurations result in usability issues. They are often disruptive, time-consuming to resolve and frustrate users and result in additional workload to IT administrators [4]. On the other hand, excessively permissive configurations results in vulnerabilities. A prominent example of excessive permissive configuration is the "Memogate" scandal, where staffers from one political party on the US Senate Judiciary Committee stole the opposing party's confidential memos from a file server that the two parties shared. This was possible in part due to an inexperienced system administrator's error in setting file permissions that provided access to unauthorised files [15].

Typically, too restrictive access control configurations are discovered by users when performing tasks and are fixed during system use. In contrast, excessive access-control configurations result in vulnerabilities that may never be detected. Additionally, what is detected could be a small percentage of the actual exploited vulnerabilities! As in the case of "Memogate", the vulnerability was exploited for over 18 months until detected after the excerpts of the stolen memos began appearing in the press and were investigated [15]. It is possible in other cases that such exploited vulnerabilities are never detected. Thus excessively permissive access control systems create vulnerable systems that may be exploited and never detected!

Why is optimal access control configuration not achieved in practice? To answer this we consider how access control rights are configured in practice.

In a typical organisation, there are three main actors that play a role in access control configuration: (1.) the *administrator* (trusted principal) who configures access control rights; (2.) the *employee* (principal) who performs authorised tasks; and (3.) typically a *manager* of the employee who authorises tasks (task authoriser) - see Figure 1.

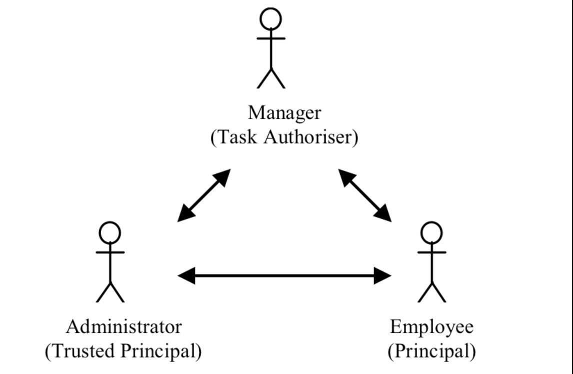


Figure 1. Actors in typical access control configuration environment

Information about authorised tasks that need to be performed by employees (principal) is sent to administrators (trusted principal) by the respective managers (task authoriser) of the employees. The administrator configures relevant access rights in the different IT systems enabling the user to perform authorised tasks.

The manager (task authoriser) informs the administrator of the authorised tasks needed by the employees/users (e.g. generate a cash flow report). The administrator is left with the non-trivial role of translating the high-level authorised tasks to low-level access control configurations in the complex mix of various IT systems and networks.

A simplified scenario is presented in Box 1 to illustrate issues pertaining to eliciting access control configuration requirements.

University Scenario:

This scenario discusses how privileges are granted to students by IT administrators in the Department of IT. The academic staff members from the Department of IT submit requirements to the IT support staff at the beginning of every semester so that the computer laboratories are configured to provide access to IT resources for students. The requirements include installation and configuration of wide ranging software on different OS platforms. These requirements are typically sent via email/web-based form to the IT support staff. Before the images for the lab machines are deployed, the parent imaged machine is available for IT academics for testing purposes.

Typically, the requirements pertaining to the specific software packages are tested and verified for smooth functioning of lab work. However, security and access control requirements are rarely scrutinized. There is a possibility for excessive permissive access configurations to exist undetected.

Consider the following instance where SQL Server (i.e. a database server), is installed on a Windows Operating System in each lab machine. The laboratory requirements may require students to create databases, users, logins for SQL databases and include backing up/restoring databases. These specifications require students to have relevant privileges on SQL Server. There are a number of possible ways to grant these privileges:

i. Option 1: Granting relevant privileges at SQL Server level to student logins/role.
ii. Option 2: Providing database administrator privileges to student logins/role which provides unrestricted access to the SQL Server database server installed locally.
iii. Option 3: Providing Windows administrator privileges on the local machine. This automatically maps as an administrator to the locally installed SQL Server in earlier versions of SQL Server.

It is clear that following option 2 (database administrator) and option 3 (Windows administrator) approaches provide excessive access control than required to perform tasks in the laboratory resulting in security vulnerabilities. Administrators however may choose option 2 or option 3 and option 3 preferably for a number of reasons.

There are a number of reasons for non-optimal access control configurations by administrators.

* *Inherent difficulty in specifying access control requirements*: Optimal access control specification requires an understanding of nature of the task, the application systems that these tasks might affect, and the different access right configurations required to perform these tasks in the different configurable systems. It is impractical to expect an administrator to be aware and consider all of these aspects in configuring access rights. Even for a skilled administrator, the possibility for human error is high in such complexity. To exacerbate, often information on tasks may be incomplete and inconsistent.

For example, in the University scenario, the academic staff members are typically logged on with administrator privileges on their office machines where they will be developing lab work, unaware of the permissions needed in completing the exercises or an idea of the privilege requirements for such work in a laboratory environment. Therefore, access requirements are not clearly specified to the IT administrators. The administrator's dilemma is then that too restrictive permissions will disrupt the lab work and generate student and staff complaints, whereas excessively permissive configurations will result in vulnerabilities.

A number of other reasons contribute to the difficulty in specifying optimal access control configurations [9]:

* *Complexity of today's systems*: With a multitude of configurable systems, hundreds of users/roles and a large set of possible permutations, it can be overwhelming for IT administrators to keep track of the different and changing access privilege configurations.

For instance, access control rights may need to be granted at various system-levels (e.g. OS level, database-level, application packages, etc.) and it is unrealistic to expect system administrators to be proficient in all these levels and systems.

* *Works fine*: Unlike other types of configurations, an excessively configured access control system may be undetected both by users and administrators, as everything seems to work fine. This makes such non-optimal access configurations harder to detect.

* *Unrewarding*: Determining optimal access control configurations requires both time and effort, but it is not demonstrable in functionality or tangible security improvements to management, especially as there are no practical means for verification.

* *Unaware of risks of excessively permissive systems*: Typically, it is internal users with malicious intent that tend to take advantage of poorly configured, excessively permissive access control configurations. The management of such incidents internally has reduced awareness of this as a risk.

* *Lack of time and effort for access control configurations*: The wide range of system support tasks required of IT administrators means that optimal access control configuration has less focus, lower priority and therefore lacks investment of their time and effort.

Today's access control configuration is, for the most part, a manual process that lacks methods and tools to conveniently and flexibly identify and configure optimal access rights in IT environments. Although default settings and configuration wizards with GUIs aim to assist in specific systems' installation and configurations, the authors are unaware of any methodology or tool that aims to provide optimal access control configurations. This paper proposes a methodology and a high-level architecture for a system to elicit and configure access control configurations. A prototype implementation of the proposed system is presented for a database server environment. The basic idea for the methodology and system was initially presented in [9].

2 Verifying Safety

Previous work on the "safety" of configurations defines a configuration as "safe" when no rights can be leaked to an unauthorised principal [7]. Unfortunately, the problem of verifying safety for an arbitrary configuration of a general access model was shown to be undecidable [7]. To overcome this problem, two approaches have been adopted: (1.) Restricting the access control model such that safety can be proven in general for that model (e.g. [1]); and (2.) Augmenting the access control models with expressions (i.e. constraints) that specify the safety requirements of any configuration (e.g. [12-13]). In the first approach, restricted models are used which are typically static and inflexible. In general, the access control policies are expressed only once by a trusted principal and fixed for the life of the system (i.e. access control policies are safe by definition). Any flexibility that may be added to these models introduces the possibility of safety problems [14]. For the second approach, a number of issues exist, including the difficulty and complexity of specifying constraint languages/models thus resulting in these constraint languages to be rarely used in practice by administrators. Also, constraints are not fail-safe, as it is possible for a missing constraint to allow unsafe configurations and possibility to add conflicts with security policies [13].

The approach presented in this paper is similar to the second approach in that no restrictions are placed on the model. Instead of constraints to verify safety, the proposed approach aims to elicit all access control rights required by users to perform authorised tasks. The proposed methodology elicits access control rights in an automated manner while users still perform tasks in a test environment, thus avoiding the need for administrators to manually determine access control rights. The configured system aims to approximate optimal access control configurations (a best-fit model).

In the proposed approach, low-level access rights are based on high-level user "tasks". Tasks are abstracted and consequently easier to specify and verify for authorisation when compared to low-level access rights. For example, a manager is able to verify that a particular financial report is allowed to be generated by a particular employee or role, rather than verifying the access rights needed for specific tables in a database or permissions to individual files and directories to perform the task. The merits in using "tasks" as a paradigm for access control and authorisation is discussed in [2,10-11]. The proposed approach is complementary, utilising tasks aimed at eliciting best-fit access control rights.

Related work on detecting and resolving too restrictive access control rights exist. Bauer et. Al [4] applied data mining to historical access control logs to identify possible missing access rights in order to eliminate misconfigurations. Although present work focuses on missing low-level access rights, the authors see future work based on detecting and verifying missing authorised high-level "tasks" as more relevant and complementary to the proposed approach.

3 Methodology

The fundamental idea underlying the proposed methodology is to elicit optimal access control configurations while users perform authorised tasks in a test environment. The test environment parallels the actual environment in that it is configured with all necessary software systems (including all access control configurable systems – which we refer to as Monitored Systems - MSs). With virtualization technologies, building such virtual test environments is more feasible than in the past. Examples of MSs include Operating Systems, Database Servers (DBMS) and application systems. In addition, Privilege Monitors (PMs) are configured for each Monitored System. PMs are programs which observe actions of each monitored system (MS) and record access control rights needed to perform these actions.

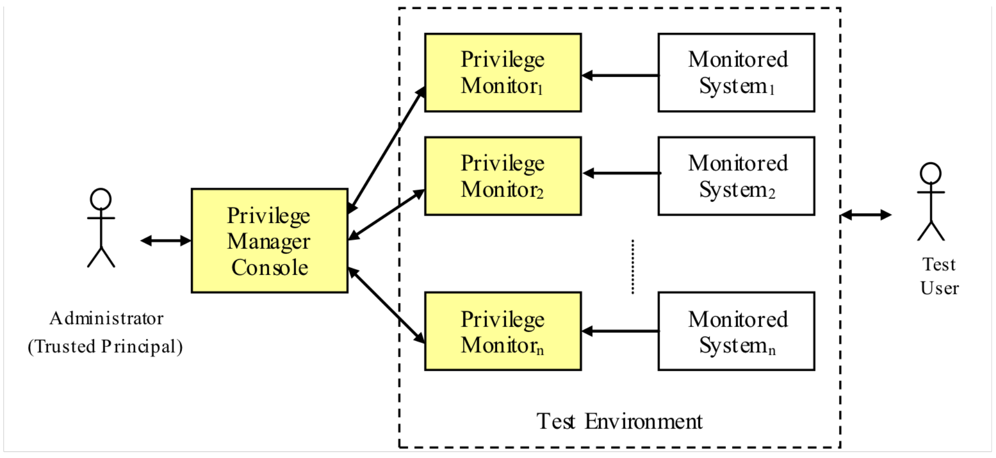


Figure 2. High-level architecture of system

The "test" user/employee performs authorised tasks. The test user is provided with unrestricted access in the test environment. The PMs observe and record all access control rights needed by MSs to complete the authorised tasks. These access rights are transmitted to the Privilege Monitor Console (PMC), which is the interface used by the administrator for evaluation of generated access control configurations, and approves them for deployment. Figure 2 illustrates the high-level architecture of the proposed system.

The steps in the process are discussed in detail below:

- *Step 1*: Initially, the test environment is configured with Monitored Systems (MSs) and Privilege Monitors (PMs).
- *Step 2*: Next, a test user (who is the task authoriser or a delegated user) from each user group/role performs authorised tasks in the test environment. The test user is provided with unrestricted access to the test environment. Providing unrestricted access in the test environment for the process of eliciting access rights does not pose a risk to the real

environment. This step is repeated for all authorised tasks of the user to generate a complete list of access control rights for the user group. PMs observe the actions for the user tasks and records the access rights needed in MSs to perform such actions.

- *Step 3*: Privilege Monitor Console (PMC) interacts with the PMs to obtain the access control rights and display them to the IT Administrator in a user-friendly, flexible manner to be reviewed and approved. As discussed later the verification process may be partially automated with constraints. At this stage, the IT administrator may add or remove additional privileges and even seek clarifications from the test user. Finally, the IT administrator approves the access control configurations for deployment in the real environment.

Note that the steps 2-3 may be repeated for different user groups/roles as needed.

The methodology elicits best-fit optimal access control configuration requirements in a flexible and convenient manner. In this approach, the manual translation of high-level tasks to low-level access control configurations is avoided! The PMs map the high-level tasks to low-level access control rights needed to perform the authorised user tasks in the different configurable IT systems.

We revisit the conditions for optimal access control configurations in order to discuss how the proposed methodology aims to satisfy them:

- *Condition 1*: *"Not too restrictive"* – ensure that all access rights to perform authorised tasks are granted.

In this approach, if *all* authorised tasks needed by users (or roles) in the real-environment have been performed in the test environment, then this process generates a complete list of access control rights satisfying condition 1.

- *Condition 2*: *"Not too unrestrictive"* – ensure that access rights for unauthorised resources are not granted.

In this approach, PMs observe actions on the configurable systems for the tasks performed in the test environment and generate access control rights to perform such actions. Therefore, access control rights for any other actions are not generated, thus aiming to be minimal. If only ***authorised*** tasks are performed in the test environment, we can safely assume that the access control rights to be restricted to perform only authorised tasks. In practice, methods to verify whether high-level tasks performed in the test environment are authorised or not is still needed. Typically, a trusted user such as the task authoriser, or a delegate of the task authoriser, may perform tasks in practice.

The notion of "access control spaces" is introduced in [14]. An access control space is the set of all possible permission assignments of a subject (or role). The three main subspaces include: (1) the permissible subspace – those assignments known to be allowed; (2) the prohibited subspace – those assignments known to be prohibited; and (3) the unknown subspace – those assignments which are neither permitted nor prohibited. The proposed approach aims to generate *all* permissible access rights for a user/role by performing all authorised tasks in the test environment. At the end of this process, a complete set of access control rights required by the user/role is generated. Thus, all access rights that are not generated through this process belong to the prohibited subspace (i.e. unauthorised tasks). The unknown subspace is eliminated. In future, if another task is deemed as authorised for the user/role, then a similar process (i.e. task-based approach in a test environment) can be used to determine access control rights required to perform such tasks and the access control rights added to the permissible subspace.

As illustrated below in the context of the University scenario, the overhead of a test environment may not necessarily be burdensome.

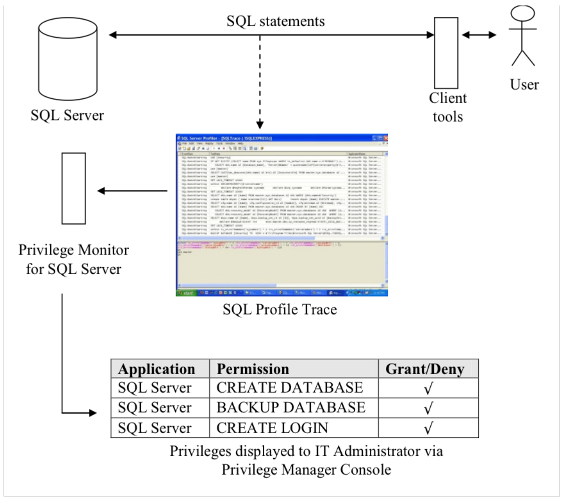
Applying methodology in the University Scenario:

In the University scenario, the lecturer installs and configures PMs in their office machines (i.e. test environment). This achieves step 1 of methodology. The lecturers enable the PMs to monitor the pre-configured systems during creation of labs materials on their office machines. Since the academic staff have administrator privileges on their own machines, this provides unlimited access in their test environment. The PMs identify and records all access control rights needed for completion of tasks (tutorials or lab exercises) in the monitored systems (step 2). Finally, when the academic staff emails/submit the software configurations for lab setup of their courses for the oncoming semester, they also attach the different PMs' output files, which consists of access control rights required by students to complete the labs using the different configured systems. The administrator is able to review the PMs' output files using the PMC, determine the access control rights needed by students in completing the labs for the semester, and also deploy them to the computer image (step 3).

The methodology can be adopted in a real-environment with minimal impact on existing processes. The academic staff need not specify access control requirements, rather PMs extract this information based on the tasks performed during the development of lab exercises. The IT administrators are able to determine optimal access control configurations to different systems required by different student groups (for courses) with minimal effort.

4 Prototype Implementation

This section discusses a prototype implementation of a Privilege Manager Console (PMC) and a Privilege Manager for SQL Server (PM for SQL Server). SQL Server [5], as a commercial Database Management System (DBMS) installable on Windows platforms, allows access control rights to be configured to both database objects and actions performed on the database. A number of reasons led to the use of SQL Server as a Monitored System in the development of the PM including (i.) familiarity with the product; (ii.) use of SQL Server for database courses in the university courses the authors are familiar with, thus providing an environment to empirically verify in a real-environment; (iii.) SQL language provides well-defined language constructs for specifying access control rights using the GRANT, REVOKE and DENY statements, thus simplifying the implementation of the PM for SQL Server and (iv.)


SQL statements
SQL Server
Client tools
User
Privilege Monitor for SQL Server
SQL Profile Trace
Privileges displayed to IT Administrator via Privilege Manager Console

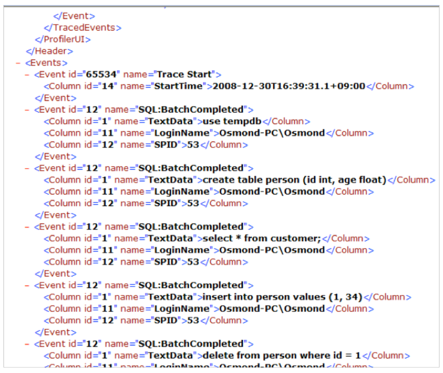
Application	Permission	Grant/Deny
SQL Server	CREATE DATABASE	√
SQL Server	BACKUP DATABASE	√
SQL Server	CREATE LOGIN	√

Figure 3. High-level flow diagram of PM for SQL Server

availability of tools such SQL Profiler [6], which enable the convenient capture of actions performed on SQL Server.

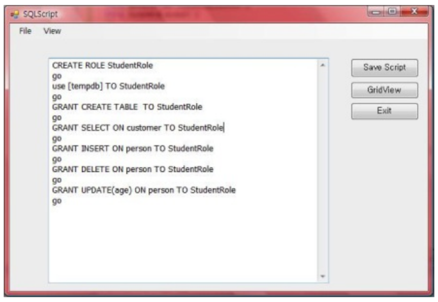
The prototype implementation of PM for SQL Server was developed as a separate module without the need to modify SQL Server database engine or its implementation. SQL Server is a client-server database server where all user actions are transmitted as Transact-SQL (T-SQL), a dialect of SQL, statements from clients to the database server. The implementation uses SQL Server Profiler [6], a tool provided with SQL Server, which can be configured to monitor T-SQL commands passed to the server. The Privilege Monitor for SQL Server is able to decipher the trace files generated by SQL Server Profiler and determine the required privileges for user actions. Figure 3 depicts a high-level flow diagram of our implementation.

PM for SQL Server reads an XML Trace file from SQL Profiler. Next, it parses T-SQL text and generates GRANT statements specifying the required access control right to perform such actions as T-SQL statements. The output of the PM for SQL Server is sent as an XML file to the PMC. The PMC then displays the access control configuration for SQL Server as a script and as a grid. Sample interfaces of the PMC are shown in Figure 4.



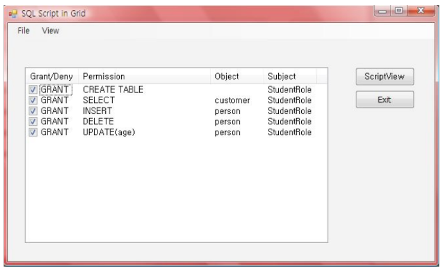
```
        </Event>
      </TracedEvents>
    </ProfilerUI>
  </Header>
- <Events>
  - <Event id="65534" name="Trace Start">
      <Column id="14" name="StartTime">2008-12-30T16:39:31.1+09:00</Column>
    </Event>
  - <Event id="12" name="SQL:BatchCompleted">
      <Column id="1" name="TextData">use tempdb</Column>
      <Column id="11" name="LoginName">Osmond-PC\Osmond</Column>
      <Column id="12" name="SPID">53</Column>
    </Event>
  - <Event id="12" name="SQL:BatchCompleted">
      <Column id="1" name="TextData">create table person (id int, age float)</Column>
      <Column id="11" name="LoginName">Osmond-PC\Osmond</Column>
      <Column id="12" name="SPID">53</Column>
    </Event>
  - <Event id="12" name="SQL:BatchCompleted">
      <Column id="1" name="TextData">select * from customer;</Column>
      <Column id="11" name="LoginName">Osmond-PC\Osmond</Column>
      <Column id="12" name="SPID">53</Column>
    </Event>
  - <Event id="12" name="SQL:BatchCompleted">
      <Column id="1" name="TextData">insert into person values (1, 34)</Column>
      <Column id="11" name="LoginName">Osmond-PC\Osmond</Column>
      <Column id="12" name="SPID">53</Column>
    </Event>
  - <Event id="12" name="SQL:BatchCompleted">
      <Column id="1" name="TextData">delete from person where id = 1</Column>
```

(a.)



```
CREATE ROLE StudentRole
go
use [tempdb] TO StudentRole
go
GRANT CREATE TABLE  TO StudentRole
go
GRANT SELECT ON customer TO StudentRole
go
GRANT INSERT ON person TO StudentRole
go
GRANT DELETE ON person TO StudentRole
go
GRANT UPDATE(age) ON person TO StudentRole
go
```

(b.)



Grant/Deny	Permission	Object	Subject
✓ GRANT	CREATE TABLE		StudentRole
✓ GRANT	SELECT	customer	StudentRole
✓ GRANT	INSERT	person	StudentRole
✓ GRANT	DELETE	person	StudentRole
✓ GRANT	UPDATE(age)	person	StudentRole

(c.)

Figure 4. (a.) XML file generated by SQL Profiler; (b.) Access control rights specified as T-SQL script; and (c.) Access control rights displayed in a grid view

Future research is intended for the development of PMs for other configurable systems (e.g. OS).

5 Discussion

A major implication of this approach is the use of tasks as a means for configuring access control rights in systems. The authors believe that, with research, development and acceptance of task-based modelling for access control configuration, manual low-level access configuration control can be minimised or even made obsolete in future. PMs and similar programs will decide access control rights based on tasks, thus eliminating the need to manually tinker with low-level access control rights.

Although the proposed approach aims to elicit optimal access control configurations, it is important to note that practically implementing PMs to generate optimal access control rights may not be straight-forward, or even possible (i.e. verifying safety in an arbitrary configuration is undecidable[7]). A simple example is used to illustrate this fact.

Example: Consider the following scenario where a manger is allowed to approve travel expenses for employees but not his/her own travel expenses. Assume that the travel expenses are maintained in the database table *TravelExpenses,* where an attribute *approverId* specifies the person who approves travel expenses and *submitterId* specifies the person submitting the travel expense claim. The relational schema of the table is shown below:

```
TravelExpense(submitterId, dateSubmitted, description, amount, dateApproved, approverId)
```

PMs may observe the task of approving a travel expense, which is an update operation on *TravelExpense* table, and may generate the access control right for the manager's tasks, which is an UPDATE operation, as follows:

```
GRANT UPDATE(approverId, dateApproved)

ON TravelExpense

TO ManagerRole
```

Note that the above statement provides access control rights for the manager to update the *approverId* and *dateApproved* columns in the *TravelExpense* table. However, it does not provide any means for disallowing approving his/her own expenses (i.e. it provides excessive access control rights). It is unreasonable to expect PMs to determine such business rules by observing user tasks in the test environment. Such business rules need to be specified as constraints in the database or application. This fact also brings out an important point: that is, the proposed methodology provides a practical approach to eliciting access control configuration requirements, but does not replace the need for constraints. While aiming to elicit "best-fit" optimal access control requirements, the proposed approach does not guarantee optimal access control configurations.

Another area where constraints are required is in the delegation of tasks to principals. It is possible that, in certain situations, access rights of tasks may be delegated to principals. Then it is important to have means to enforce "task safety"; that is, tasks are not delegated to unauthorised principals. Similar to constraints in [12], where safety of configurations are verified by constraint expression, the authors perceive a need for constraint verification for task safety. Future work on *task safety and* constraint languages is needed.

The applicability of the proposed technique is not limited to access control configurations in a stand-alone environment but in many other areas (e.g. collaborative environment [9]). Such work is left for future investigation.

6 Conclusion

In this paper, the severity of access control configurations was discussed. In practice, access control configurations are rarely configured optimally, resulting in either excessively permissive systems or fewer access rights configured than required. The latter reason causes usability issues while the former cause vulnerabilities.

Although optimal access control configuration is elusive (i.e. verifying “safety” for an arbitrary configuration which was proven to be undecidable [7]), there aren’t any “practical” means (i.e. methods and tools) for approximating optimal access control configurations. Thus, access control configuration remains to be a manual task performed by IT administrators. This paper presented a practical methodology and architecture for a system that aims to elicit access control configurations automatically based on tasks performed by users in a test environment. A high-level architecture and prototype implementation of the proposed system for a database environment was presented. A university scenario was used to illustrate the applicability of the methodology.

A major implication of the proposed methodology is the use of tasks to determine access control rights. The authors believe that further work in this area can lead to minimising or making obsolete the manual low-level access control configurations currently performed by administrators.

Future research directions include development of PMs for complex configurable access control systems (e.g. OSs) and developing easy to use environments for administrators to flexibly configure and manage permissions across heterogeneous IT systems.

Acknowledgements

This research was partly funded under The University of Newcastle’s - Faculty of Science and IT’s Undergraduate Summer Scholarship program 2008.

7 References

[1] D. Bell and L. L. Padula, "Secure computer systems: Mathematical Foundations," Vol. 1. Tech. Rep. ESD-TR-73-278. Mitre Corporation, 1973.

[2] K. Irwin, T. Yu, and W. H. Winsborough, "Enforcing security properties in task-based systems," in *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*. Estes Park, CO, USA: ACM, 2008, pp. 41-50.

[3] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson, "CSI/FBI Computer Crime and Security Survey," Computer Security Institute 2005.

[4] L. Bauer, S. Garriss, and M. K. Reiter, "Detecting and Resolving Policy Misconfigurations in Access-Control Systems," in *Proceedings of the ACM Symposium on Access Control Models and Technologies*. Estes Park, CO, USA: ACM, 2008, pp. 185-194

[5] Microsoft Corporation’s SQL Server, "SQL Server Home Page", Microsoft Corporation. [Online] Available: http://www.microsoft.com/sqlserver.

[6] Microsoft Corporation’s SQL Server Profiler, "SQL Server Profiler", Microsoft Corporation. [Online] Available: http://msdn.microsoft.com/en-us/library/ms173757.aspx

[7] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, pp. 461-471, 1976.

[8] NSF Workshop on Assurable and Usable Security Configuration, Fairfax, Virginia, USA, Aug 11-12, 2008. Available: http://www.safeconfig.org/.

[9] R. Athauda, G. Skinner, and B. Regan, "A Methodology to Minimise Excessively Permissive Security Configurations," in *Proceedings of the 8th WSEAS International Conference on Applied Computer Science* (*ACS'08*). Venice, Italy: WSEAS Press, 2008, pp. 187-192.

[10]R. K. Thomas and R. S. Sandhu, "Towards a task-based paradigm for flexible and adaptable access control in distributed applications," in *Sixth National Computer Security Conference*. Baltimore, MD, USA: ACM, 1993, pp. 138-142.

[11]R. K. Thomas and R. S. Sandhu, "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management," in *Proceedings of the IFIP WG11.3 Workshop on Database Security*. Lake Tahoe: Chapman & Hall, 1997, pp. 166-181.

[12]R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, pp. 38-47, 1996.

[13]T. Jaeger and J. E. Tidswell, "Practical safety in flexible access control models," *ACM Transactions on Information and System Security*, vol. 4, pp. 158-190, 2001

[14]T. Jaeger, X. Zhang, and A. Edwards, "Policy management using access control spaces," *ACM Transactions on Information and System Security*, vol. 6, pp. 327-364, 2003.

[15]U. S. Senate Sergeant at Arms, "Report on the investigation into improper access to the Senate Judiciary Committee's Computer System," U.S. Senate March 4, 2004.