Simplifying Enterprise Wide Authorization Management Through Distribution of Concerns and Responsibilities

SHARIL TUMIN
University of Bergen
IT Dept
P.O. Box 7800, 5020 Bergen
NORWAY
edpst@it.uib.no

SYLVIA ENCHEVA
Stord/Haugesund University College
Faculty of Technology, Business and Maritime Sciences
Bjørnsonsg. 45, 5528 Haugesund
NORWAY
sbe@hsh.no

Abstract: Authentication lets a system know who you are, while authorization controls your resources access rights and what operations you are allow to perform. Resources have owners to whom the resources belong to. The owner knows best who is allowed to access her resources at any one time. Distribution of concerns and responsibilities can be effectively used for efficient management of enterprise wide authorization. A collection of users with similar rights to a resource can be logically grouped. Managing user groups, recourse groups, access relationships and permissions is tedious due to complex inter-relationships between actors and among actions. A Web-based application which implements a higher level abstraction of access management that mappes specific low level operating system access control can be deployed to access local manager to manage their resource access effectively and economically, which in turn will increase the system's security level of the entire organization.

Key–Words: Authentication, Authorization, Distributed Management, Delegation, Enterprise IT security, Multi-tiers Web-based System

1 Introduction

Personal computers have no special need for authentication and authorization. The owner of a PC has the sole rights on its resources. Multi-users systems are of a different category. These systems have to serve multiple users and these users will usually be assigned to different roles. Permissions assigned to different users for different resources will vary and much will depend on what role a particular person has in an organization [3]. Roles are dynamic properties that people in an organization have. Some of these roles are transient in nature and yet others are more permanent. Authorization is also used as a mechanism that provides privacies protection for different system's users.

System administrators or system managers in a multi-users system environment have to manage different *resource-user-permission* relationships (*access relations*) and guarantee users' privileges and protect users' privacies. Privacy and privilege are two contradicting values which an authorization management needs to address delicately. The question of *who*, *what*, *when* and *why* in relation to authorization can be at time a daunting task for system managers to perform.

Authorization follows authentication in a system security implementation. While authentication controls system login, authorization controls resources access and permissions. Resources have definite ownership relationships, in terms of users ownerships and groups ownerships. Managing authorization is tedious due to complex relationships between ownerships, resources and permissions. A group defined as a collection of users or resources can be used to reduce the number of existing relationships and thus ease authorization management tasks.

It is natural that authorization management of resources falls on the hands of resource owners through delegation. Managing authorization using a Web based management application will provide a higher level abstraction which can facilitate implementation of such distributed management framework. Managers and users a like, will be assisted with an effective and economical management tools, which, in turn will increase the system's security level.

Privacy and privilege are two opposite poles of access control management where a delicate balancing acts, as any computer systems managers have to perform.

The problem lies within the scope of these complex *who-what-when-why* relationships. Typical questions would be,

1. why this user needs access to that file;
2. for what purpose;

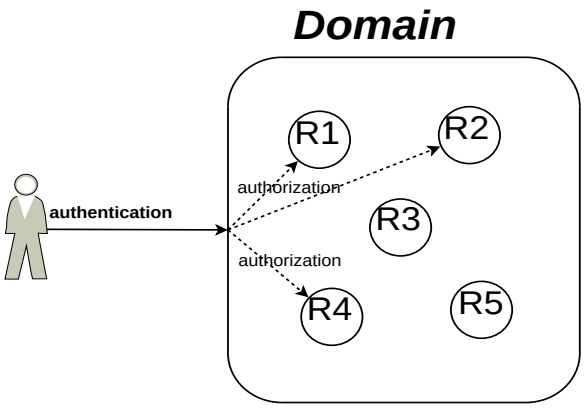

Figure 1: Authentication vs Authorization

3. by which permissions; and

4. for how long period of time.

Usually the system managers are in the dark to what was the original purpose of that specific resource. The original owners would best know the initial purpose of the resource. Given that, the data concerning *what-why* can be stored in a database during an initial creation process of a particular resource, however, the details of *who-what-when* can be problematic. Since a shared resource was created for a specific purpose, therefore it is most natural to give the responsibility of access management to the original owners of the resource. These delegations of responsibilities at the enterprise level can provide efficiency in the system management process at lower operative system levels.

Another important issue in authorization management is controlling the effective time period of *access relations*. Without an automatic cancellation of an *access relationship*, a user might gain unwarranted access beyond the justified time period prescribed by her roles and her organizational duties. If this remains unchecked it will definitely lead to possibilities for compromising the operational and privacy policies of the organization in terms of lost of private data and unauthorized execution of critical processes, done by unwanted individuals internal or external of the system. An automatic cancellation of *access relationships* is an important security function. It will surely help to mitigate the negative effects of overdue authorizations.

To reduce the number of combinatorial *who-what* combinations, both users and resources can be arranged into groups. Instead of relating an individual user to a particular resource, the system presented in this paper will define *access relationships* by relating groups of users to groups of resources. This technique will ease the burden of authorization management but at the same time increases the implementation difficulties. A user-group members can change over time and similarly will resource-groups. User-groups membership management is practically easy to implement, for example by using LDAP group scheme. However, resource-groups management can be difficult to do in practice as resources are managed at an individual resource level by the operation system and not as a collection of differently located resources.

To implement such a system, we assume that the enterprise has an **IdM** (Identity Management System) in which personal data about all users, users' credentials and users' default authorizations are stored. Together with these authentication and default authorization data, an additional system for implementing operational authorization data such as user-groups, resource-groups, groups memberships and *access relationships* are also needed. These information can be stored as a part of the IdM's database or can be implemented as a subsystem to the IdM. In this paper we propose a Web-based system for distributed authorization management through delegation - **DAMD**. The subsystem model of **DAMD** will provide the most flexible solution in serving an enterprise authorization management endeavour.

2 Background

2.1 Authentication versus Authorization

Authorization follows authentication as shown in Figure 1. Authentication is but a small part of a security mechanism that implements access control on system resources.

Authentication is a process by which the system asks you to identify yourself and to give proof that you are indeed who you said you are. There are two critical information concerning authentication that the system shares with you, namely your user identification and your password. You and only you know your password.

The system stores your password in an encrypted form, usually in a one-way cryptographic hash format, for example crypt, MD5, or SHA. During authentication process, the identity and the password given by you will be checked against an identity database, for example /etc/passwd file, LDAP (Lightweight Directory Access Protocol), AD (Active Directory) or SQL (Structured Query Language) database user table. If the hashed password and the identity pair given earlier matches the pair stored in the identity database then you are authenticated.

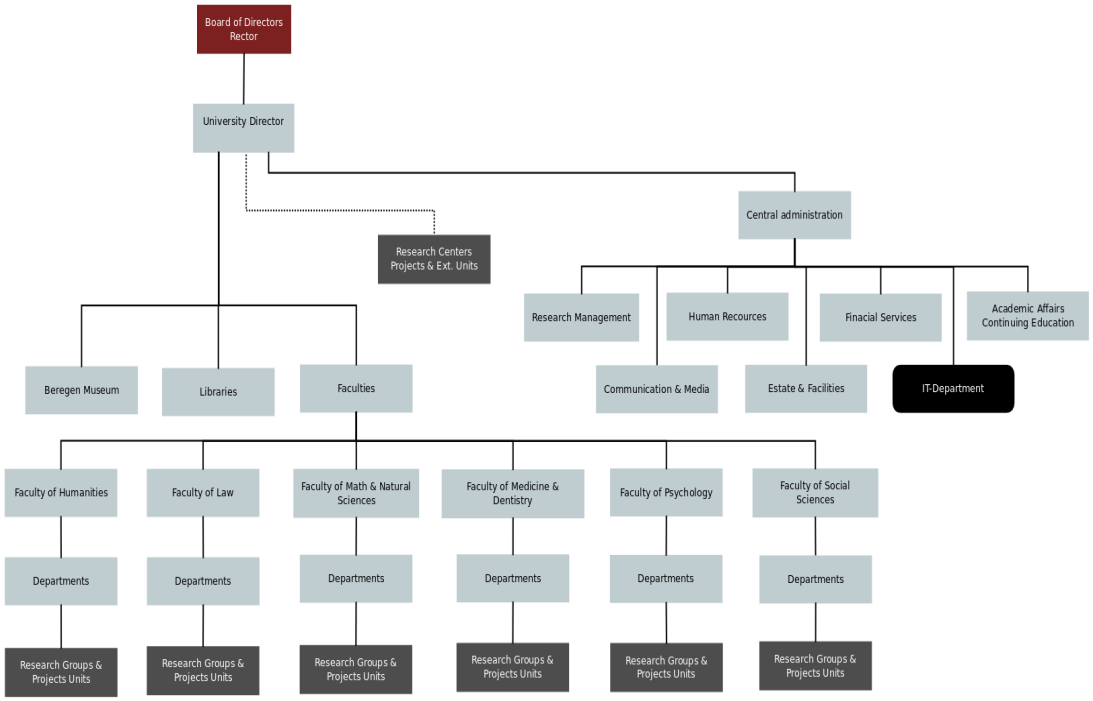

Figure 2: A Typical Organizational Map of a University

Once a user is authenticated, the system will then know who the person is. Related to this particular person are her roles, access rights to system resources and permissions on these resources, technically speaking, her authorization.

Authorization is a process by which the system assigns permissions to authenticated users on system resources. The permissions are usually checked at the moment a user wants to perform certain action on a resource, for example read or write. There are many users and many resources in a system. To relate each individual user to each individual resource is wasteful efforts. What is usually being practiced is to assume no access rights and no permission unless specifically stated in the system. A resource has a definite ownership, thus the owner and the resource owned are implicitly related. The management of these relationships of ownership, access right and permissions between users and resources can be very tedious depending on the number of users, resources and types of permission.

To reduce number of users and resources relationships, groups are introduced into the authorization management. A group of users (*usergroup*) is defined as a collection of users having similar access rights and permissions on a resource or a group of resources. While, a group of resources (*resourcegroup*) is defined as a collection of resources for a unit of a workgroup for a particular organizational task.

Typically an organization is organized in a hierarchical structure, and typically is represented as an inverted tree as shown in Figure 2, showing an organization structure of a university. Under each faculty are departments. Under each department are divisions. A *resourcegroup* can either be defined as faculty wide, department wide or division wide. Thus, for a faculty wide *resourcegroup* the *usergroup* associated with it will be all users belonging to the faculty and similarly for departmental and divisional defined *resourcegroup*. One thing to remember is that a *resourcegroup* does not include hierarchical definition upward the tree path. A *resourcegroup* is only defined on the level at which it was defined.

Certain users-resources relationships can be automatically managed using the organizational structure, for standard resources for example department common shared storage areas. Whereres, *resourcegroup* defined for a specific task on a specific level need to be managed individually using associated *usergroup*.

It is very natural that authorization management of a *resourcegroup* falls on the hands of the owners. This can be achieved through delegation. The own-

ers are best to know what the resources are used for and who are illegible to access them in order to performe a specific task. Managing authorization using a Web based application will provide a higher level of abstraction which can facilitate the implementation of such distributed management framework.

2.2 Concepts as Basic Structures

Concepts as basic structures of logic are most important for turning information into knowledge because humans first grasp realities by concepts wherefore concepts are the basic units of thought and knowledge, [19].

Definition 1 *[2] Let P be a non-empty ordered set. If* $sup\{x, y\}$ *and* $inf\{x, y\}$ *exist for all* $x, y \in P$*, then P is called a* lattice.

In a lattice illustrating partial ordering of knowledge values, the logical conjunction is identified with the meet operation and the logical disjunction with the join operation.

Definition 2 *[18] A* context *is a triple* (G, M, I) *where G and M are sets and* $I \subset G \times M$*. The elements of G and M are called* objects *and* attributes *respectively.*

For $A \subseteq G$ and $B \subseteq M$, define

$$A' = \{m \in M \mid (\forall g \in A)\ \ gIm\},$$

$$B' = \{g \in G \mid (\forall m \in B)\ \ gIm\}$$

where A' is the set of attributes common to all the objects in A and B' is the set of objects possessing the attributes in B.

Definition 3 *[18] A* concept *of the context* (G, M, I) *is defined to be a pair* (A, B) *where*

$$A \subseteq G,\ \ B \subseteq M,\ \ A' = B \ \ and \ \ B' = A.$$

The *extent* of the concept (A, B) is A while its *intent* is B. A subset A of G is the extent of some concept if and only if $A'' = A$ in which case the unique concept of the which A is an extent is (A, A'). The corresponding statement applies to those subsets $B \in M$ which is the intent of some concepts.

The set of all concepts of the context (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$.

Definition 4 *[18]* $\langle \mathfrak{B}(G, M, I); \leq \rangle$ *is a complete lattice and it is known as the* concept lattice *of the context* (G, M, I).

There is no need for authorization if all users to have every and equal rights to system resources. Authentication is enough to control system access and disallow others to access from outside of the system. However, resources and permissions to resources are given differently to different users. Organizational, roles, and jobs structures of an organization are also reflected on how *access relations* are defined in the organizational ICT (Information and Communication Technology) system.

An enterprise ICT system is comprised of many subsystems, with different authorization strategies related to operative systems in used. Some are more complicated and involved than the others. In this paper we focus on clasical authorization strategies as implemented in Linux/Unix multi-users and multi-hosts ICT system [5], [6]. By multi-users, we mean that the **IdM** needs many user credentials for authentication and the **DAMD** subsystem needs to manage dynamically many users' groups and memberships tightly coupled with *access relationships* for authorization at a higher level than the one provided by the individual operating system.

Authorization processes and information access permissions are discussed in [13], [17], [16], and [15].

A user-accounts database containing all user credentials, normally username and password pairs, together with authorization and permissions database are necessary parts of a multi-user system. Authentication data is important to system protection from outside intrusions. System protection can be improved by pro-actively checking for weak passwords [10]. Authorization data is important for system integrity, users' privacies and system security from within. Good authorization management policies and practices can contribute to the betterment of an ICT system as a whole, a fact that not many system managers consider as a priority task within security management work.

The listing below shows a typical files listing on a Linux/Unix work station. Each line of the listing is preceded by a '`d`' for a directory or a '`-`' for a file and permission flags patterns of `rwxrwxrwx` where a '`-`' means the permission is set to off. The first '`rwx`' block from the left signifies the *owner* (**u**) permission to that particular file or directory. The second block signifies the *group* (**g**) permission and the last block is for the *others* (**o**) (all the rest). The '`rwx`' stands for *read*, *write* and *execute* respectively.

```
drwxrwxrwx  5 sharil sharil     4096 2009-04-06 13:51 tmp
-rw-r--r--  1 sharil sharil 44459788 2009-12-01 12:38 D80.ps
drwx------  3 sharil sharil     4096 2009-03-31 13:52 .dbus
drwxr-xr-x  2 sharil sharil     4096 2009-11-18 14:17 Documents
-rw-------  1 sharil sharil       16 2009-03-31 13:52 .esd_auth
drwx------  2 sharil sharil     4096 2009-03-31 15:49 .tsclient
```

Each line of the listing is preceded by a permission flags patterns of `rwxrwxrwx` where a '-' means the permission is set to off. The first '`rwx`' block from the left signifies the *owner* (**u**) permission to that particular file or directory. The second block signifies the *group* (**g**) permission and the last block is for the *others* (**o**) (all the rest). The '`rwx`' stands for *read*, *write* and *execute* respectively.

The most relevant for us here is the group permission block. Setting control permissions to the middle block and group ownership of the resource will provide a simple yet effective authorization management at the file system level. Without a management system at a higher level, owners and system managers tend to employ the (**o**) permission block. Using the (**o**) will give people access on a particular resource regardlessly, but a more selective access is usually needed when security and privacy issues are taken into consideration.

Collecting users with similar roles and rights into groups, and similarly done to resources, makes a lot of sense. In what follows, we show examples for seven user and nine resources, and by capitalizing on visualization produced using concept lattices [8] we can easily see that the complexity of *access relations* can be reduced significantly by grouping.

Table 1: Users to Resources

u/r	r1	r2	r3	r4	r5	r6	r7	r8	r9
u1	*		*		*				*
u2		*		*					
u3				*			*	*	
u4	*		*		*				*
u5		*				*	*		
u6				*			*	*	
u7		*				*	*		

Table 1 shows a simple example of *access relations* between these users and resources, and Figure 3 shows its concept lattice. A limited amount of users and resources is chosen in the example for the sake of clarity. A concept lattice [4] can be used to analyze any number of *access relationships*. However, increasing the number of *access relationships* can make the corresponding concept lattices difficult to read.

From Figure 3, we can see that resources $\{r1, r3, r5, r9\}$ are shared by two users $\{u1, u4\}$, resources $\{r4, r7, r8\}$ shared by users $\{u3, u6\}$, and users $\{u5, u7\}$ are sharing $\{r2, r6, r7\}$. Therefore the possibilities for potential grouping are clearly shown by the lattice.

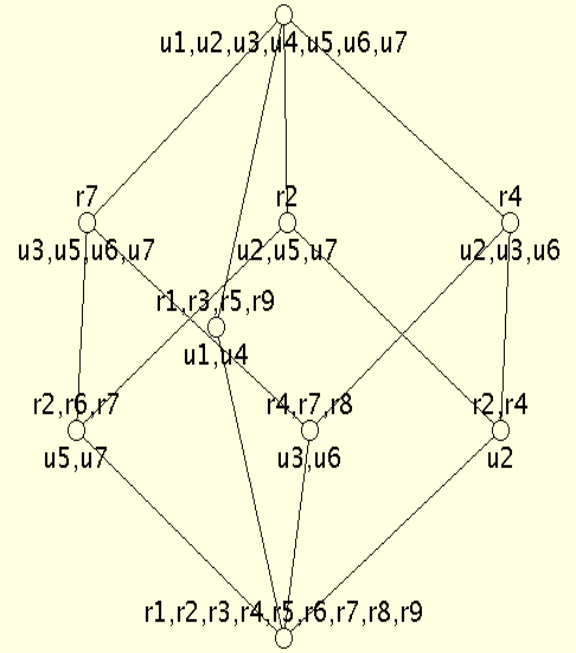

Figure 3: Users-Resources Relationships

We can group the users and resources in a number of ways. The most natural grouping scheme (α-scheme) would be to group all users using a particular resource, one group for each resource: $g1 = \{u1, u4\}$, $g2 = \{u2, u5, u7\}$, $g3 = \{u1, u4\}$, $g4 = \{u2, u3, u6\}$, $g5 = \{u1, u4\}$, $g6 = \{u5, u7\}$, $g7 = \{u3, u5, u6, u7\}$, $g8 = \{u3, u6\}$, $g9 = \{u1, u4\}$.

Grouping users in this way will produced a trivial *access relations* as shown in Table 2.

Table 2: Groups to Resources

g/r	r1	r2	r3	r4	r5	r6	r7	r8	r9
g1	*								
g2		*							
g3			*						
g4				*					
g5					*				
g6						*			
g7							*		
g8								*	
g9									*

From the *access relations* shown in Table 2 a simple concept lattice is produced, Figure 4. When comparing the lattices shown in Figure 3 and Figure 4, the

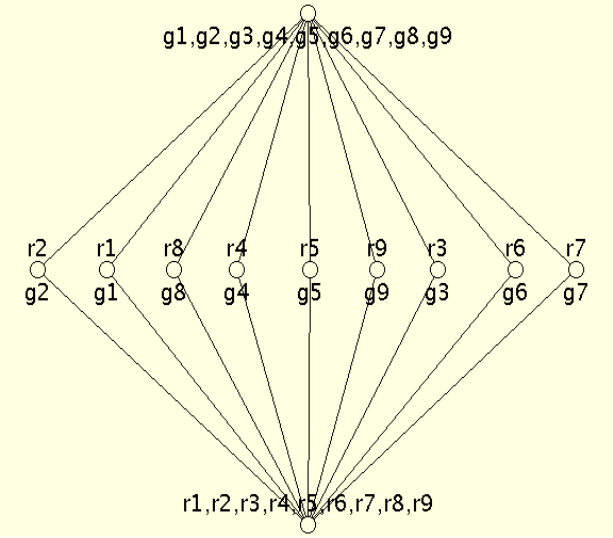

Figure 4: Groups-Resources Relationships

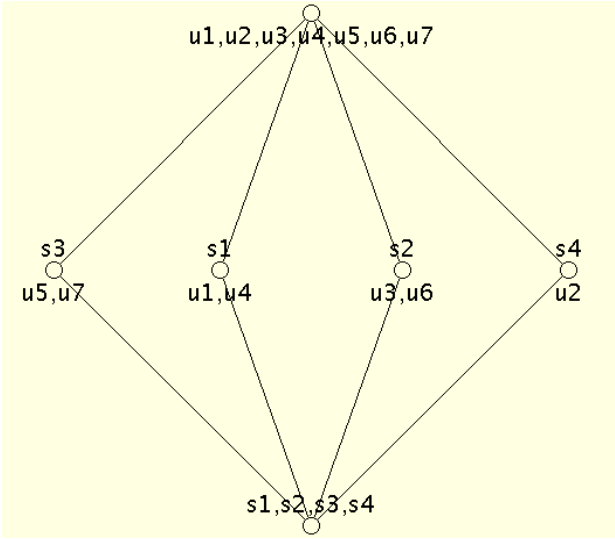

Figure 5: Users-Resources Groups Relationships

later lattice has a much simpler structure. Therefore grouping evidently reduces the complexity of *access relations*.

A different grouping scheme (β-scheme) is also considered to be useful. In this case the groupings are done on collections of resources defined as follows: $s1 = \{r1, r3, r5, r9\}$, $s2 = \{r4, r7, r8\}$, $s3 = \{r2, r6, r7\}$, $s4 = \{r2, r4\}$.

The difference between α-scheme and β-scheme is that, while referring to Table 1, the α-scheme grouping are done vertically while the β-scheme are done horizontally across the *access relations* table.

Table 3: Users to Resources Groups

u/s	s1	s2	s3	s4
u1	*			
u2				*
u3		*		
u4	*			
u5			*	
u6		*		
u7			*	

For this particular example, the β-scheme produces a simpler structure with four groups compare to nine groups for α-scheme. Depending on how the *access relationships* are to be deployed either α-scheme or β-scheme can be used. However, due to how the operating system implements access control, α-scheme is easier to implement for a single resource. The β-scheme is used to implement access control on a group of distributed resources.

There is a severe limitation in the way traditional Linux/Unix access control is implemented. A **g** control block is bound to only a single group owner. So, this fact will make it difficult to share the same resource to different resource groups $\mathbb{S}_i$ and $\mathbb{S}_j$ since the relation $\mathbb{S}_i \cap \mathbb{S}_j = \emptyset, i \neq j$ must be true at all time. One solution is to make copies of resources whenever they are shared to different groups. If

$$\mathbb{S}_i \cap \mathbb{S}_j = \{r_k\}, i \neq j$$

then

$$\mathbb{S}_i = \mathbb{S}_i \cap (\mathbb{S}_i \ominus \mathbb{S}_j) \cup \{r_{ki}\}$$

and

$$\mathbb{S}_j = \mathbb{S}_j \cap (\mathbb{S}_i \ominus \mathbb{S}_j) \cup \{r_{kj}\}.$$

This workaround will work well if the resources are read only and static. For dynamically changing resources an elaborate tracking and revision control sub-system needs to be supported as a part of the system.

A better way is of course to use the more modern *acl* (Access Control Lists) whenever the operating system has the support for it. By using the *setfacl* and *getfacl* commands, any groups other then the group owner can be given access permission to a particular resource.

Let $\mathbb{A}$ and $\mathbb{B}$ be collections of authentication and authorization data respectively. $\mathbb{A}$ contains all user

credentials and $\mathbb{B}$ contains all definitions for resources access control. $\mathbb{A} = \{(u_i, p_i)\}$, $i = [1, n]$, where (u_i, p_i) is the username and password pair for user i. $u_i \in \mathbb{U}$ where $\mathbb{U}$ contains all enterprise users. $\mathbb{B} = \{(r_j, g_j, P_j, C_j)\}$, $j = [1, m]$, where (r_j, g_i, P_j, C_j) is a resource, group, permissions and constrains tuple for enterprise resource j.

$r_j \in \mathbb{S}_j$ where $\mathbb{S}_j$ is a resource group,
$\mathbb{S}_j \subseteq \mathbb{R}$ where $\mathbb{R}$ contains all resources,
$g_j \in \mathbb{G}$ where $\mathbb{G}$ contains all defined groups,
$P_j \in \mathbb{P}$ where $\mathbb{P}$ contains all defined permissions,
$C_j \in \mathbb{C}$ where $\mathbb{C}$ contains all defined constrains, then, $u_i \in g_j, \bigcup_{j=1}^{m} g_j \subseteq \mathbb{U}$ and, $r_i \in \mathbb{S}_j, \bigcup_{j=1}^{m} \mathbb{S}_j \subseteq \mathbb{R}$.

If $\mathbb{A}$ is compromised then $\mathbb{B}$ will loose much of its security functions. Suppose $u_1 \in g_1$ and $g_1 \times r_1$. If u_1's credential was stolen then access control defined by *access relations* of $g_1 \times r_1$ will be compromised.

If the compromised user u_1 is known and it is not the *root* user, excluding u_1 from all groups will reduce the effect of an internal security violation or a security breach performed by external agents. The ability to quickly act in group membership exclusion procedure is an important security function.

The proposed solution should be able to handle a short term notice for massive removal risky *access relations* in respond to a security incident. In normal day-to-day authorization management, the proposed system must be able to provide administration framework that is able to support a medium scale concurrently management activities done by normal users acting as administrators due to delegation responsibilities. The main considerations in decreasing order of importance will be, 1) security; 2) usability; 3) scalability; 4) reliability; and 5) efficiency.

3 System

As explained in Section 2, Linux/Unix based systems make use of the **u**, **g**, and **o** permissions blocks to control access and permissions for the *owner*, *group* and *other* respectively, on a particular resources. The most interesting control block for us here is the **g**.

Any of the three control block can be manipulated by using `chmod` operative system command. By using an octal mode number the permissions flags can be switched on and off. As an example, we will vary the **g** and keep all permissions for **u** and **o** blocks constant as shown in Table 4.

By using mode number `0000` the access of a particular resource to all ordinary users regardless of ownership or group membership will be disabled. While `chmod` and mode numbers are useful to implement permissions control, the group associated with the resource is important to us as mechanism for implementing an α-scheme and a β-scheme mentioned in Section 2.

Table 4: chmod permissions mode

Mode (octal)	Permissions
0770	-rwxrwx---
0760	-rwxrw----
0750	-rwxr-x---
0740	-rwxr-----
0730	-rwx-wx---
0720	-rwx-w----
0710	-rwx--x---
0700	-rwx------

In the α-scheme, one group identifier is associated with one resource. This is useful for the case of one executable program to be shared among a group of users and to none others. However, sharing clusters or groups of resources is more common. In the β-scheme, a group identifier is associated to a group of resources. Thus in α-scheme, $u_i \in g_j :: r_k$, while in the β-scheme, $u_i \in g_j :: \mathbb{S}_j = \{r_k\}, k = [1, m]$.

In the proposed system, a resource is a directory own by a particular user with a particular group identifier associated with it. The owner is given a full permission on that directory so that she can create new sub-directories or files. If these new sub-directories and files will be shared by group members then these resources must be associated with the group of the parent directory. It is recommended that all **o** permissions are disabled.

The responsibility of creating and setting up group associations and permissions is delegated to the current owner of the parent resource in an operating system level. Moreover, this user will also be responsible in the users membership management of the associated group by using Web-based management application. Through this Web-based application the "delegated manager" can give and take away other users membership to the associated group.

3.1 System Data Model

As the system implements its authorization management of owner, root directory resource and group association, it is natural that these data should be collected into a central database. Here we have proposed four basic database tables; 1) *delegated_managers*, 2) *groups*, 3) *resources* and 4) *members*. The detail entity-relationship diagram is shown in Figure 6.

The *delegated_managers* table contains owners of the root directory resources. These *delegated man-*

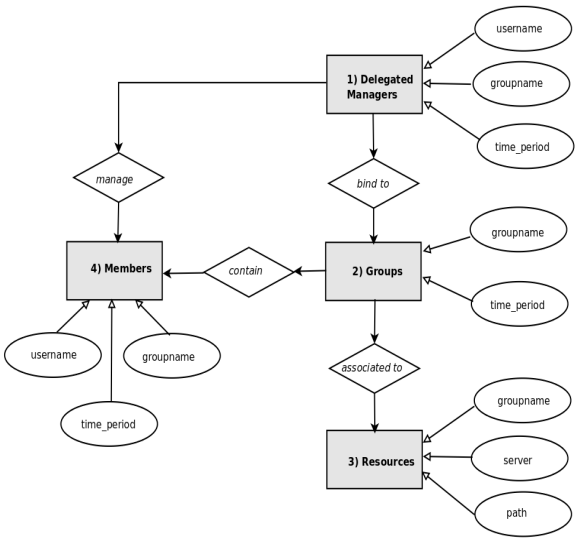

Figure 6: Entity-relationship Model for Delegation

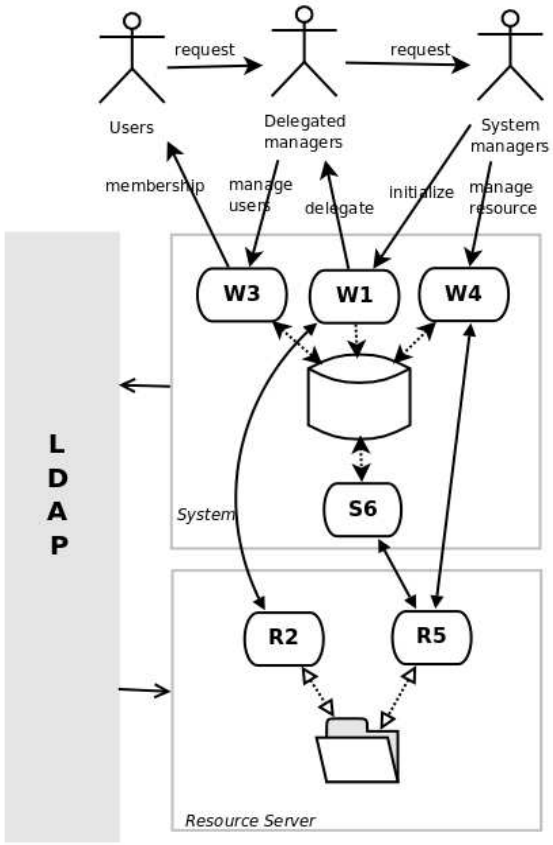

Figure 7: Process Flow Diagram

agers are responsible in maintaining the proper permissions and group associations to all files and subdirectories under a particular root directory resource. The *groups* table defines all groups in the system to be used to create resource-group associations stored in the *resources* table. Groups memberships are defined in the *members* table.

3.2 System Components

Building on top of these data models, the proposed system will support these operational functionalities. These functions are of three types; 1) Web-based applications (**W***), 2) RPC-based, a remote procedure call applications installed on each servers (**R***), and (**S***) scheduled application invoke from system for automatic clean-up jobs.

This applications conceptual process flow diagram is shown in Figure 7 and it is summarized as follows:

W1 - System managers' Web interface for initializing the data tables for a new *access relation*. The important initial data are the owner if the resource, a unique group identifier, the path of the resource, the server name, and the time period of this particular *access relation*. This in fact will introduce a new *delegated_managers*, *groups*, *resources*, and *members*. The first member of the group is the resource owner. This application will trigger the execution of **R2** on the resource's server side using RPC (remote procedure call).

R2 - This RPC server application on a resource server will create (`mkdir`) for the resource at a particular location, give the resource its proper ownership (`chown`) and set the initialize (`chmod`) permissions of `0770`, i.e. full access to user owner and group owner.

W3 - A Web interface for delegated managers for managing; 1) other delegated managers for a particular *access relation*, and 2) group membership to this particular resource associated group. A delegated manager can share the management responsibility with other delegated managers. The effective period will be automatically controlled by the time period if given. A user is given right to access a particular resource when she is a member of the resource associated group. The membership is recorded in the *members* as well as in the group scheme at the LDAP server. The operational access control is done on the level of operation systems using data found in that LDAP server. The effective

membership period will be automatically controlled by the time period if given.

W4 - A Web interface for system managers for the administration of a group resources associated with a particular group. A new resource can be added to the group by populating the *resources* table and invoking **R2**. A resource can be deleted from the group by removing its entry from *resources* and invoking **R5** on the resource server. System managers can also; 1) disable, 2) enable, and 3) delete *access relation* as a whole. The process of disabling an *access relation* does not remove users from a group membership but setting all resources permissions to `0000`. Deleting an *access relation* means deleting all related data in the database and LDAP, and deleting all related resources from the servers.

R5 - This collection of RPC will; 1) disable, 2) enable, and 3) delete access to resources on the level of operating system. The disable and enable procedures will use `chmod` or *setfacl* commands. The delete procedure will permanently remove resources and will use the `remove` to delete file resources and `removedirs` to recursively delete directory and sub-directories.

S6 - These collection of scheduled applications will; 1) remove users' membership from groups, 2) remove users from delegated managers rights and responsibility, and 3) remove *access relations* from system database and servers clean-up jobs. All these activities are triggered at the end of specified time periods. Since remove *access relations* can effect working environment of many users, the process of removing a particular *access relation* will be proceeded by sending email warnings to delegated managers first. The resources in a particular *access relation* will be disabled first prior to the actual deletion at a later prescribed date. These scheduled programs will invoke procedures defined in **R5**.

Suppose a successful attack on an enterprise is discovered. Depending on the security risk evaluations of the current threat, system administrators can choose to disable suspected users' authorization by 1) removing resources permissions; and 2) removing group memberships.

Disabling users' authorizations can be done in two different ways; 1) by changing permissions on a particular resource using `chmod 0000`, and 2) by revoking users' membership of the group associated with the resource.

Since in practice we can not disable users' enterprise passwords, alternatives 2) and 3) provide us with a nice compromise. We will then be able to warn and communicate with users with enterprise email addresses. For a time being their access to shared resources will be blocked. Users will be asked to change their passwords. The accounts that do not conform to this will be disabled. After all the suspected resources are tested and investigated and found to be free from malicious programs, the authentications will be restored to their original status.

3.3 System Implementation

The proposed system will be implemented as a multitiers Web-based application, built using free and open source software. The users using their Web browsers will interact with an Apache [11] Web server using HTTPS (Hypertext Transfer Protocol Secure) communication protocol. The programmable environment and the middlewares are written in Phython [14]. The back-end database is implemented using PostgreSQL [12]. The Python programming environment is used as integrating middleware between the Web server front-end and the database back-end.

Some of the important Python packages used to implement the system are:

1. *mod_python* - programmable module to Apache,
2. *xmlrpclib* - XML (Extensible Markup Language) RPC (Remote Procedure Call),
3. *tlslite* - SSL v3 (Secure Sockets Layer) and TLS v1 (Transport Layer Security) libraries,
4. *pyPgSQL* - API (application programming interface) to PostgreSQL and 5) standard Python libraries for example - *os*.

4 Conclusion

Newer distribution of Linux/Unix support `acl`, can be used to define a more fine-grained discretionary access rights for files and directories are available. Using `acl`, a resource can be associated with multiple groups on the operating system level. The proposed system can accommodate both mechanisms for access control. With `acl`, resources can be shared by many groups at group level and by many users at user level. However, managing authorization on user-resource relations is not recommended since such practice will complicate management tasks.

By splitting the management data in the database and operational data in LDAP, we managed to implement authorization management by delegation. This will greatly reduce the burden of system managers since the authorization management tasks on shared resources are done by the original resource owners.

Having a centralized database for authorization adds values to the whole management tasks, namely; 1) some automatic mechanism can be employed to remove overdue membership and even delete a whole *access relation* with its related resources and associated group in a timely manner; 2) another important possibility is to be able to lock down a group of resources temporarily in the events of a security breach; and 3) The *what-why* for each shared resource or resource-group can be documented.

Authorization management by delegation improves security since it provides a mechanism by which users themselves can manage resource sharing without the need to resort to the use of **o** permissions block. The system is a Web-based application, therefore is scalable and usable. Modular designed and loosely coupled sub-systems provide reliability. In operational mode, the system provide efficient access control since only LDAP data is effected during a group membership modification.

References:

[1] Alghathbar K., Mahmoud H. A., *Block-Based Motion Estimation Analysis for Lip Reading User Authentication Systems*, WSEAS Transactions on Information Sciece and Application, Vol 5 (6), 2009, pp. 829-838.

[2] Davey, B. A., and Priestley, H.A.: Introduction to lattices and order. Cambridge University Press, Cambridge, (2005)

[3] Ferraiolo, D., Kuhn., D. R., and Chandramouli R., *Role-Based Access Control* Artech House, Computer Security Series, 2003.

[4] Ganter, B., Wille, R., *Formal Concept Analysis - Mathematical Foundations* Springer, Heidelberg, 1999.

[5] Evi Nemeth, Garth Snyder, and Trent R. Hein, *Linux Administration Handbook (2nd Edition)* Prentice Hall, Pearson Education, 2006.

[6] Evi Nemeth, Garth Snyder, Scott Seebass, and Trent Hein, *UNIX System Administration Handbook (3rd Edition)* Prentice Hall, 2000.

[7] Dictionary, Encyclopedia and Thesaurus & The Free Dictionary, *http://www.thefreedictionary.com/*, 2009 (last accessed).

[8] Maarten Janssen, Online Java Lattice Building Application for Concept Lattices, *http://maarten.janssenweb.net/jalaba/JaLaBA.pl/*, 2009 (last accessed).

[9] Herve' Morizat, The 2008 International Information System Security Survey. *http://www.devoteam.com/images/File/SecuritySurvey-BD.pdf*, 2009 (last accessed).

[10] Matt Bishop, Daniel V. Klien, *Improving system security via proactive password checking Computer & Security*, Elsevier Science Ltd., 14 (1995) 233-249 1995.

[11] Apache, The Apache Software Foundation, HTTP server, *http://www.apache.org/*, 2009 (last accessed).

[12] PostgreSQL, The world's most advanced open source database, *http://www.postgresql.org/*, 2009 (last accessed).

[13] Pranata I., Skinner G., *Digital Ecosystem Access Control Management*, WSEAS Transactions on Information Science and Application, Vol. 6 (6), 2009, pp. 926-935.

[14] Python Programming Language, *http://www.python.org/*, 2009 (last accessed).

[15] Skinner, G., *Making A CASE for PACE: Components of the Combined Authentication Scheme Encapsulation for a Privacy Augmented Collaborative Environment*, WSEAS Transactions on Computers, Issue 1, Volume 7, January 2008.

[16] Skinner, G., *A Privacy Augmented Collaborative Environment (PACE)*, Proceedings of the 7th WSEAS International Conference on Applied Computer Science (ACS07), Venice, Italy, November 21-23, 2007.

[17] Skinner, G., *Shield Privacy: A conceptual framework for Information Privacy and Data Access Controls*, WSEAS Transactions on Computers, Issue 6, Volume 5, June 2006, pp. 1375-1384.

[18] R. Wille, Concept lattices and conceptual knowledge systems, Computers Math. Applications, 23(6-9), 1992, 493–515

[19] R. Wille, Why can concept lattices support knowledge discovery in databases?, Journal of Experimental and Theoretical Artificial Intelligence, 14 2 & 3 (2002), 81-92