Role Based Access Control: Adoption and Implementation in the Developing World

By
Loy A.K. Muhwezi

Master's Thesis in Computer Science
Thesis number:

Supervised By
Dr. Martijn Oostdijk

Radboud University Nijmegen
Institute for Computing and Information Sciences (ICIS)
Security of Systems

August 2007

Abstract

Role based access control (RBAC) is a recent access control approach that provides flexibility and ease of privileges' management through the concept of roles. It addresses the rigidity found in mandatory access control (MAC) and the Trojan horse problem common in discretionary access control (DAC). Although the level of research in access control and RBAC in particular is high in the developed world, little has been done in the developing world. More so, most research findings are presented using complex methods that are hard to understand by users in such countries. In this research, a different approach to investigate access control based on systems in a financial institution in Uganda (an example of the developing world) is taken. The research findings are presented in a simple way to meet a special need by users in such parts of the world, to understand and address concerns that hinder adoption and implementation of access control systems. The challenges faced by the institution in its bid to control access to its information systems are presented, and the issues which need improvements are also discussed. The basics of setting up an access control system, RBAC in an organization are presented, and by use of the Solaris 9 operating system (OS) RBAC features, the benefits the mechanism offers are analyzed. Recommendations and some remedies to issues identified are given, and the direction of future research is also presented.

Acknowledgements

I would like to thank Dr. Martijn Oostdijk for all his guidance and support that made this thesis possible. I wish to thank the entire management and staff of the financial Institution in Uganda especially its MIS department, for all the support rendered to me during the case study. I am grateful to Dr. Erik Poll for reading this thesis. To the entire staff of ICIS and External Relations office (Radboud University), I am grateful for all the services rendered to me during my stay in The Netherlands. I am also thankful to all persons whose word of encouragement assisted me to complete my studies successfully. Special thanks to my fellow students from Uganda, Prof. Theo van der Weide, Nicole el Moustakim, Ger Paulussen, Paula and Marijke. I wish to thank NUFFIC and the Faculty of Computing and Information Technology (Makerere University), for the opportunity given to me to develop my research skills. My warmest thanks go to my family for their encouragement, love and support. Bernard, without you it would have been hard to persevere to the end.
Above all, glory to the Almighty God for the strength and blessings to accomplish the entire Masters course in time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security of information systems and access control mechanisms are increasingly becoming crucial in the twenty-first century as computing systems and network technologies evolve with new threats and risks. Both developed and developing countries are equally affected, as globalization becomes the order of the day. To cope up with the trend that security is taking, new access control approaches such as role based access control (RBAC) have gained popularity to solve issues that were left hanging by traditional approaches like mandatory access control (MAC) and discretionary access control (DAC).

In our research, we investigated information security with regard to implementation of security patterns during system development. RBAC as one of the access control patterns embedded in systems, was chosen and used in a case study to understanding the way access control systems are adopted and implemented in the developing world. However, characterizing countries in a developing-country-scale is beyond the scope of this research. This thesis report discusses access control issues in general and RBAC in particular, using both literature study as well as case study approach. We have tried to keep it simple baring in mind the targeted part of the world in our current and future research, but at the same time highlighted major areas that require attention.

1.1 Motivation

From initial literature study, we learnt that security patterns are increasingly being written and used in system development but often such systems are manipulated or illegally accessed and information is lost. However, we observed that, security has improved to some extent where security patterns are embraced in system development, and are well implemented and utilized. A lot of literature we came across, explicitly talks about use of security patterns in form of access control mechanisms in system development to enhance security. We noted that, their adoption and implementation in developing countries is still limited which calls for more research. And this thesis attempts to address that.

This research focuses on evaluating what is said in literature using a real life case study to answer the following general and specific questions respectively: How can security patterns be used to evaluate security in a system that uses private networks?

How does Role Based Access Control (RBAC) pattern provide security to a system?
This specific question is divided into sub questions for ease of handling.

1. What is Access Control and what is RBAC?
2. How does RBAC differ from older approaches (Mandatory Access Control and Discretionary Access Control) and Access Control Lists?
3. How is RBAC implemented?
4. How does RBAC work?
5. What are the benefits and limitations of using RBAC?

1.2 Security Patterns

Security patterns were first proposed by Yonder and Barcalow [48] for the software community to address recurring security problems. Since then, a lot has been published on security patterns and many of them have been written. Schumacher*et al* [42] give an overview of security patterns and describe them as structured documents that capture security expertise in the form of solutions to recurring security problems. The essence is to help system developers understand security and plan for it in the infancy of system development.

Safe guarding systems of any kind during design and development stages by use of security patterns [[20], [42], [44], [48]], have been proposed, and are already being implemented by system developers. Among security patterns that have attracted attention are access control patterns, role-based access control (RBAC) inclusive.

There are many options to describe elements of security pattern's structure according to the context at hand and the experience of the pattern writer. However, each pattern must at least have the following:

- a name which is easy to remember and gives a picture of what the pattern is all about,
- a problem in a context to describe security issues addressed by the pattern,
- forces to elaborate major aspects of the problem,
- a solution to describe an appropriate design rule to resolve these forces,
- related patterns since patterns do not work in isolation, and
- known uses

1.3 Information Security Policy

An information security policy strengthens the security and well being of information resources. It acts as a base of information security within an organization. Information security is concerned with safeguarding information from unauthorized access (accidental or intentional), modification, destruction, or disclosure. Adequate security exhibits the following properties:

- Confidentiality - preventing unauthorized disclosure of information.
- Integrity - preventing unauthorized modification of information.
- Authentication - Ensuring that users seeking access to information are what they claim to be.

The policy should be developed, implemented and managed properly to meet these security requirements and the needs of the organization. However, most organizations in the developing world do not have information security policies in place. Even organizations which have the policies tend to be less vigilant in implementing them.

In addition to the general information security policy, an access control policy is essential. An access control policy helps to define and document business and security requirements that access controls must meet.

1.4 Access Control

Access control is a process of limiting right of entry or/and use of objects (data/information and resources of an information system) to only authorized subjects (users, programs, processes, devices, and other system resources requesting access) by enforcing specified rules, on objects' use. The first line of security defense is to control access to an information system. In the bid to provide this defense, many organizations tend to mind a lot about physical security especially limiting access of outsiders. While it is good to guard against outside attacks, existing literature [18] show that higher percentages of successful attacks come from within the organization as a result of accidental or malicious attacks by employees.

Increased use of networked computer systems and the Internet has made access control an essential requirement for security. In the past, physical security was enough for paper documents and individual computers. As documents are increasingly being stored and shared on networked systems, access control becomes a crucial issue. With computer-based access controls, who or what accesses a specific system and by what means is specified. And there should be an appropriate way of specifying it.

Hu *et al* [22], advises organizations planning to implement an access control system to consider three concepts: access control policies, models and mechanisms. An access control policy can be described as a high level guideline that presents a collection of rules which specify what is allowed and what is not, on information objects. Models formalize access control policies and state how they work. Mechanisms enforce access control policies by defining the low level functions that implement the controls stated in the formal model.

Common access control policies are Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC). DAC policies are based on an individual owner or creator of an object defining access rights for its users, and a user is capable of sharing those rights with another user. MAC policies are based on a central authority defining access rights and no individual user can change them. RBAC introduces the role concept that captures permissions, users and objects. Access is then based on the administrator assigning permissions to roles using the principle of least

privileges. DAC policies are commonly implemented using access control matrix model, while MAC policies are commonly implemented using multilevel security (MLS) and Bell-LaPadula (BLP) models. RBAC is policy neutral and has a variety of models, with a Reference Model presented in [15].

1.5 Role-Based Access Control

Role based access control (RBAC) emerged as a superior alternative to traditional approaches (MAC and DAC). While different access control approaches may have different capabilities, research done on RBAC [[13], [14], [16], [22], [27], [34], [41], [42]], highlights its high ability to handle security compared to traditional approaches. The essential concept of RBAC is that privileges are assigned centrally through roles which make it able to restrain users from having discretionary access to objects, and at the same time allowing flexibility. In RBAC, permissions are granted to job functions (Roles) other than individual users. Individuals instead assume roles assigned to them by the organization, according to the descriptions of their job(s) or tasks they perform. Roles are easy to use because they are relatively stable, while users and permissions may change rapidly.

Ferraiolo *et al* [15] present the RBAC Reference Model which attempts to standardize RBAC models. It consists of four components - Core RBAC, Hierarchical RBAC, Static Constrained RBAC and Dynamic Constrained RBAC. Core RBAC defines the basic features (users, roles, permissions, operations, and objects) that are found in any RBAC system. Hierarchical RBAC adds role hierarchy to Core RBAC model to cater for increased numbers of roles. Since some job functions (roles) in organizations overlap with each other, and at times with users having similar sets of permissions on some objects, roles can be represented in a hierarchy. Constrained RBAC enhances core and Hierarchical RBAC with Static Separation of Duty (SSOD) and Dynamic Separation of Duty (DSOD) constraints.

In commercial environments, preventing disclosure, unauthorized modification, fraud and error is important, but easy maintainability of the authorization database to avoid errors and security administrative costs is very necessary. The RBAC's flexibility and capability to handle most of the deficiencies in MAC and DAC, such as Trojan horse problems, policy dependency and some rigidities, have made it popular in commercial enterprises. Its support for the principles of least privileges and separation of duty (SOD) makes it even more powerful. These features make RBAC a better approach and give it an advantage over MAC and DAC to provide an efficient and less complicated means of maintaining the authorization database. Moreover, some RBAC implementations can strongly simulate MAC [[28], [36]] while others can simulate DAC [39], depending on the purpose of the system.

1.6 Case Study

In the literature study phase, we realized that information security mechanisms in general and access control in particular is evaluated basing on surveys and implementations done in the developed World. Even surveys and case studies

that may be beneficial to the developing world, in most cases are conducted on-line [5],in which these countries rarely participate. Standards, technologies, and mechanisms related to access control and other security mechanisms are less understood and adopted in the developing world. The case study investigated security of systems in a financial institution based in a developing country - Uganda.

Our inspiration is that, there must be issues that need to be understood and resolved, which hinder adoption, development and deployment of access control technologies in the developing world. The following questions among others, guided our study:

- What are the access control policies, technologies and mechanisms in place and what influences them?
- What are the methods of work and implementations of existing technologies in local organizations in the developing world?
- What are the benefits and limitations of these technologies?
- What are the future plans as far as these technologies are concerned?

While developing systems with security in mind is important, some organizations just buy ready to use systems without understanding how they work. Embedding security strategies in the system is one good thing, but utilizing the strategies properly to enhance security is another issue.

The case study revealed that, there are many factors which may influence the way security issues and access control in particular are handled in such parts of the world. We observed lack of appropriate knowledge and expertise as major factors among others. Other factors are: organizations having much trust in physical security, organizational structure and culture, information being presented in complex ways for novice users, poverty, and limited access to information, to mention a few. An oversight is also observed on the part of the system developers, they do not consider developing systems with mechanisms that alert customers to configure all security features at the time of installation.

In the case study, an RBAC implementation plan is made, and the Solaris 9 Operating System (OS) RBAC features are utilized by making root a role. Kienzle *et al* [26] present security patterns temperate that discuses consequences of a security pattern in terms of security properties. In a similar manner, the consequences of Solaris RBAC root configuration are also discussed along those lines. An improvement in system security is observed.

Despite all the benefits of RBAC, its implementation is difficult and costly, and a proper plan is required. The Information Technology (IT) team has to understand how RBAC works and compile information on Systems, Hardware and Software to determine their level of security required basing on the core mission of the organization. It also requires full support of top management of the organization especially the Human Resource (HR) department to identify job functions of users basing on their job descriptions. If it is a new access control system it requires educating employees to understand and embrace the system.

1.7 Organization of the Thesis Report

The remainder of the report is organized as follows:

- Chapter 2 discusses the access control policies and mechanisms, providing the general overview and the implementation of each of the traditional access control policies, MAC and DAC.
- Chapter 3 provides the overview of RBAC and its implementation as an access control mechanism as well as a security pattern.
- Chapter 4 presents the case study on access control mechanisms' implementation issues in the developing world and discusses the implementation of the Solaris 9 OS RBAC. It also presents case study results and recommendations.
- Chapter 6 summarizes the benefits of RBAC and some improvements required, presents conclusions and future work.

Chapter 2

Access Control

Increased use of networked computer systems and the Internet has made access control an essential requirement for security. In the past, physical security was enough for paper documents and individual computers. As documents are increasingly being stored and shared on networked systems, access control becomes a crucial issue. Access control tries to enforce security by limiting access to objects.

This chapter defines access control in Section 2.1 and discusses access control policies in Section 2.2. Access control mechanisms are discussed in Section 2.3 and the chapter is concluded in Section 2.4. The discussion in this chapter forms the basis of answering questions one and two for this research. It acts as a platform for comparison of role based access control (RBAC) with other approaches and for its choice for security evaluation in our case study.

2.1 Definition

Access control can be viewed from different perspectives depending on the environment it is discussed. Access control is defined in many ways by different writers. Section 2.1.1 presents related definitions of access control we came across during our literature study. Section 2.1.2 presents the definition that is used in this research.

2.1.1 Related Definitions

According to Sandhu and Samarati [40] access control is concerned with limiting the activity of legitimate users, but relies on and coexists with other security services in a computer system, as summarized in Figure 2.1, adopted from Sandhu and Samarati [40]. During authentication, credentials for a user are validated. Authorization follows authentication and is expressed in terms of access rights or access modes. A reference monitor provides the necessary properties for the access control system, by comparing information stored in the authorization database with what the user has provided during authentication before the control mechanism decides to grant or refuse access. Separation of authorization and access control is hard, and sometimes these notions are mistakenly used

interchangeably. The incidents that take place in authentication, authorization up to accessing objects are recorded in auditing. Other related definitions are:

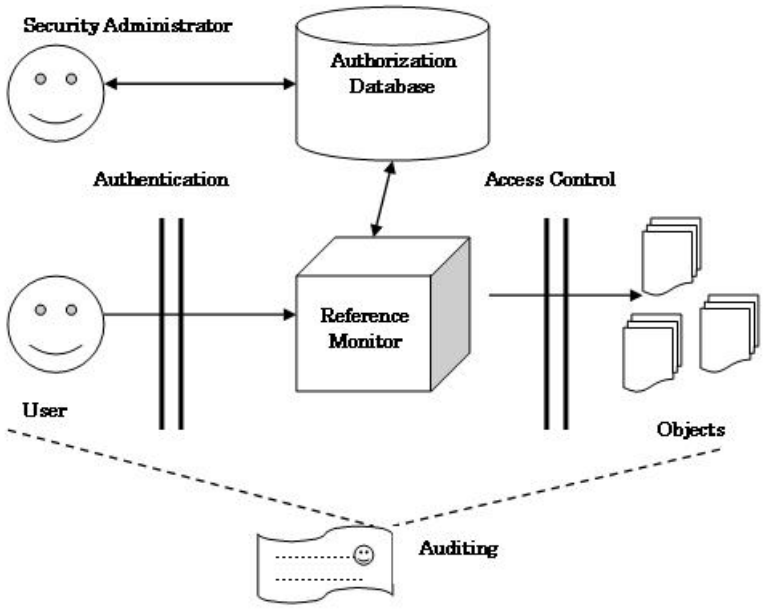

Figure 2.1: Access control and other security services

- Access control is concerned with determining the allowed activities of legitimate users, mediating every attempt by a user to access a resource in the system [22].
- Access control is the ability to permit or deny the use of something by someone [46].
- Access control is generally referred to as restricting access based on something other than the identity of the user [3].
- Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied [17]
- Access control is to enable authorized use of a resource while preventing unauthorized use, or use in unauthorized manner [45].

A set of other definitions of access control presented in [19] is below:

- A service feature or technique used to permit or deny use of the components of a communication system
- A technique used to define or restrict the rights of individuals or application programs to obtain data from, or place data onto, a storage device.
- The definition or restriction of the rights of individuals or application programs to obtain data from, or place data into, a storage device.

- Limiting access to information system resources only to authorized users, programs, processes, or other systems.
- That function performed by the resource controller that allocates system resources to satisfy user requests.

2.1.2 Our definition

Definitions above have something in common, and rotate around preventing unauthorized access and ensuring authorized use of objects based on defined rules. We do not dispute most of the definitions above, but we agree with Sandhu and Samarati [40].

Let us take an example of restricted access to watch a football match in a real world situation with regard to other services that coexist with access control. What one does is to buy a ticket to present at the entrance for authentication as a rightful spectator for the match. At this point, the spectator is allowed in, but does not sit in any pavilion other than that authorized for, by the ticket or gate pass. With the right ticket (may be by colour or number, or gate pass) the spectator is then allowed access to a seat in the right pavilion. The auditing part would be for the management to keep track of all attempts that have happened to violate rules during the whole process. These audit controls would be useful to analyze spectators' behavior for future matches. This gives us a general understanding of what happens in controlling access.

With the definitions in Section 2.1.1, and our general understanding, the following definition shall hold for this research.

Definition 2.1 *Access control is a process of limiting right of entry or/and use of objects (data/information and resources of an information system) to only authorized subjects (users, programs, processes, devices, and other system resources requesting access) by enforcing specified rules on objects' use.*

Hu *et al* [22], advises organizations planning to implement an access control system to consider three concepts: access control policies, models and mechanisms. An access control policy can be described as a high level guideline that presents a collection of rules which specifies what is allowed and what is not, on information objects. Models formalize access control policies and state how they work. Mechanisms enforce access control policies by defining the low level functions that implement the controls stated in the formal model. The subsequent sections, discuss common access control policies, models and mechanisms.

2.2 Access Control Policies

Access control implementation decisions on information assets are based on the organization's policy. An organization may specify an access control policy for its information and information resources basing on the level of security protection required. It may categorize organizational information objects according to the level of sensitivity, may relate information to particular unit categories or a combination of categories, or even distinguishing them according to the types of access and action to be performed on them.

This section gives an overview of access control policies in general in Section 2.2.1. Section 2.2.2 presents mandatory (MAC) policies, discretionary (DAC)

policies are presented in Section 2.2.3, and role-based access control (RBAC) in Section 2.2.4.

2.2.1 Overview

Access control policies define rules and conditions applied to a requesting entity for accessing a resource or executing operations exposed by an application or a service [44]. A simple description especially for the developing world to understand an access control policy is presented in [22], as high-level requirements that specify how access is managed and who may access information under what circumstances. Access control policies are categorized as discretionary (DAC) and Non-DAC (NDAC), with MAC and RBAC being the commonly known NDAC policies.
According to Hu *et al* [22] concepts commonly used by most access control policies and mechanisms, and which are useful for understanding access control are as follows:

- *Object*: An entity that contains or receives information. These include records, files, fields, pages, directories, processes, programs, network devices and any other data container or moving device.
- *Subject*: An active entity, generally in form of a person, user processor, methods, system process or device that causes information to flow among objects or change the system state. In some cases, a subject can be used interchangeably with a user.
- *Operation*: An active process invoked by a subject. Basically an operation is a way to access an object in form of an action.
- *Permission (privilege or right)*: An authorization to perform some operation on the system. Specifies how a subject is allowed to access an object. Permissions may determine specific access rights such as whether a user can write to, read from, or execute an object.

2.2.2 Mandatory Access Control Policies

Hu *et al* [22] describe Mandatory Access Control (MAC) as one in which access control decisions are made by a central authority, not by the individual owner of the object. System security policy has full control and users may not change access rights to their resources other than specified by an administrator. MAC can also be defined as a system of access control that assigns security labels or classifications to system resources and allows access only to entities (people, processes, devices) with distinct levels of authorization or clearance [29].

The controls are mandatory because labelling of information happens centrally. Controls may be based on rules set for example through classification levels on objects and subjects. Figure 2.2 extends the general access control service presented in Figure 2.1, to include MAC policy. The labels or/and rules on objects and subjects are stored together with the permissions on those objects in the authorization database for comparison whenever a request is made. The whole concept of MAC originated from military security where confidentiality of classified information was a top goal. The military security model enforced

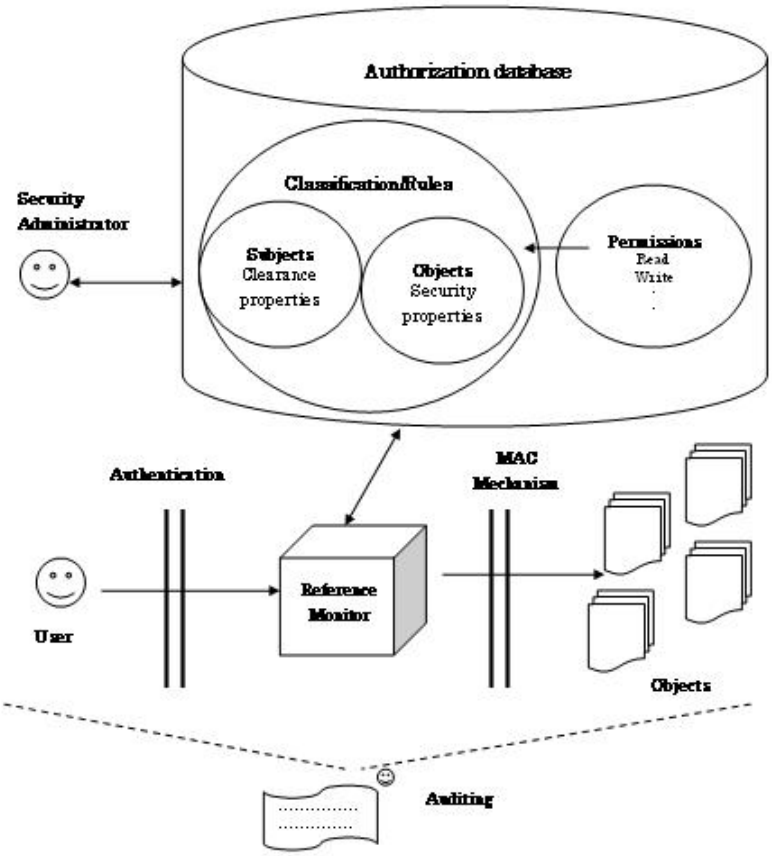

Figure 2.2: MAC policy

the principle of least privilege by use of a security classification hierarchy on objects and subjects. Examples of popular implementations that support MAC policies are multilevel security models, and other models for commercial sector presented in this section. This section also discuses the maintainability of the authorization database in MAC.

Multilevel Security

Multilevel Security (MLS) is an example of implementing a MAC policy. It is commonly used in military and intelligence community to process various security levels of classified information. MLS assigns security levels in the form of classifications to all objects, a security clearance to each subject, and ensures that all subjects only have access to objects which they are cleared to access. Each object and each subject has a label, and the label is compared to privileges assigned to it before access is granted. Security levels such as Top Secret, Secret, Confidential and Unclassified in some models are given to objects.

Bell-LaPadula: The Bell-LaPadula (BLP) confidentiality model is described by Hu *et al* [22] as a multilevel security model used to formally specify MAC policy. The BLP model enforces MLS by implementing the simple security property (no read up) and the *-property (no write down). Access is granted by the system to the requesting subject after the clearance label has been compared to the classification label. More elaborate versions of BLP involve compartments in addition to security levels, but here we have kept it simple. Compartments are additional markings (such as area of operation) added onto a classified material for further restricted distribution. In BLP, access control is based on comparing the labels and applying the 'simple security property' and the '*-property'.

In Figure 2.3, privileges (read(r) and write(w)) on files, f_1, f_2, f_3 and f_4 enforced by the BLP-MAC policy would be as follows:

$priv(f_1) = \{(Anna : r, w), (Bob : w), (Char : w), (Dan : w)\}$
$priv(f_2) = \{(Anna : r), (Bob : r, w), (Char : w), (Dan : w)\}$
$priv(f_3) = \{(Anna : r), (Bob : r), (Char : r, w), (Dan : w)\}$
$priv(f_4) = \{(Anna : r), (Bob : r), (Char : r), (Dan : r, w)\}$

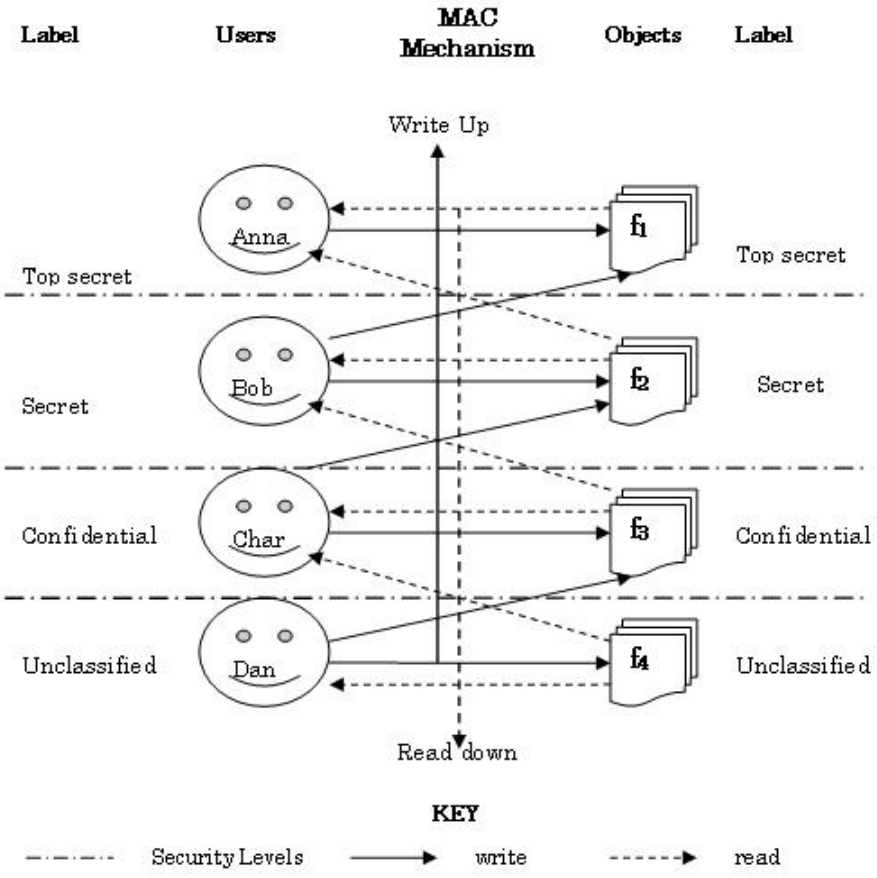

Figure 2.3: MAC Policy - Simple Bell-LaPadula model

In the BLP model, if a subject S has a clearance level l_s, can only read from object (file) F, with a classification level l_f, if and only if $l_f \leq l_s$, where

$l_i < l_i + 1$, and can only write to F, if and only if $l_s \leq l_f$. In deciding who is to read, write or execute which file, the authority examines who needs what according to the user's clearance level of operation.

However, the BLP model could not address the integrity of data, though it well satisfied the secrecy requirement of military operations. Another drawback was the ability of a malicious user corrupting a file while writing up. Jaeger *et al* [23] discusses another problem of information flow through covert channels.

Biba integrity model: The Biba integrity model is another example of a multilevel security model to formalize a MAC policy that was developed by Kenneth J. Biba in 1977 [47]. The Biba model enhances BLP, specifically on data integrity issues. Objects and subjects are grouped into controlled levels of integrity. The properties used by BLP are instead defined in a reversed way. The simple security property in the Biba model is 'no read down' and the *-property is 'no write up'.

Nevertheless, very few commercial operating systems support BLP and Biba models because of their rigid classifications of data. This rigidity makes it hard to publish public information by organizations implementing this kind of MAC. Take an example of an unalterable (unchangeable) public document, it would have to exist at a level below unclassified so that no one writes to it.

Other Access Control Models

Many other access control models, other than BLP and Biba models have been in use to support MAC policies, and some are presented in this section. For more details on these models, the reader is referred to the references indicated.

Clark-Wilson model: The Clark-Wilson model [10] was developed to address security issues in commercial environments. It is primarily concerned with the integrity of data which is often an important goal in commercial data processing. It is based on two principles: separation of duty and well-formed transactions. In well-formed transaction concept, a user should not manipulate data randomly, but only in constrained ways that preserve integrity of data. Separation of duty principle ensures that at least two people are required to cause a change in the set of well-formed transactions. This model generally implements MAC policies but in a different approach from military model by emphasizing integrity other than secrecy.

Chinese wall model: The Chinese wall model was proposed by Brewer and Nash [7] to prevent information flow that causes a conflict of interest. It requires that users are only allowed access to information which is not held to conflict with any other information that may be already in their possession. When a user chooses a dataset to access on a computer, any other request to access another dataset which conflicts with that accessed before, is denied. In other words a Chinese wall is created for that specific user around the datasets. Brewer and Nash describe the two conflicting datasets as being on 'the wrong side' of the wall. The user can only access any other dataset in a different conflict of interest class, but which is included in the wall as soon as it is accessed. The Chinese wall policy combines commercial discretion with legally enforceable mandatory

controls and is meant to help financial institutions. The restrictions on users' access based on the history of their previous accesses, make this model not flexible for commercial environments. It also does not provide access control on the level of individual objects or tasks. More models were still required to address all these issues and yet provide necessary confidentiality and integrity.

Maintainability of Authorization Database

In BLP model, storing classification labels on all objects and clearance levels on all subjects for comparison is required. Every object and subject in a system, must have a label, which are compared according to simple security property and *property.

Definition 2.2 *A privilege p viewed as a unit of measure for access rights, is a pair (O, m) that can be granted to a subject S by the administrator, through an access control label, L*

Where,

O is a set of a unique name or identifier of an object and its label

m is the access mode e.g read, write

S is a unique name for a user or a user's process, and

L is the security label (l_o or l_s) associated to access properties, where l_o and l_s are labels for object and subject respectively.

Definition 2.3 *Let a set of authorizations Auth(S) for a given user S who joins an enterprise be defined and stored in the authorization database as:* $Auth(S) = l_s, p_s$

Where, $p_s = \{p_1, ., p_n\} = \{(O_1, m_1), ., (O_n, m_n)\}$ is a set of privileges on all n objects, which S is authorized to access, and l_s is the clearance label assigned to S.

If an enterprise is big with a large number of objects which subject S has to access, then n will be big and hence the number of authorizations to be stored. Suppose the number of individual users also increases, the authorizations will also be affected in the same way since every object the user is to access has to be updated. The database would become so large that its maintainability would almost be impossible. Every time a change occurs to an individual object or subject, the administrator has to make changes to authorization data. This makes it hard to implement a security policy in large organizations with big numbers of employees. Maintaining up to date information is hard and errors are likely to occur.

Trojan Horse Problem

Because of rigidities in MAC, the Trojan horse problem is not as common as with DAC. For this regard, most systems implementing DAC use MAC mechanisms in addition to prevent Trojan horse vulnerabilities. However, some processes or programs in MAC, must be allowed to violate the simple security property or the *.property to be able to accomplish their tasks. For example, in a financial institution a program may be allowed to access confidential data to be able to produce a summary that is not confidential for public consumption. This program may be regarded as a trusted subject designated for that purpose and

not subjected to 'write up' and 'read down' rules. Nevertheless, a malicious program pretending to be the trusted one, can still make it possible for Trojan horse vulnerability without notice.

Suppose Anna has read access to department (Dept) X objects, and read and write access to Dept Y objects and Bob has read access to Dept X objects and Dept Z objects. A Trojan horse program running with Anna's privileges can read Dept Y objects and write to Dept Z objects, hence enabling Bob to read Dept Y objects.

2.2.3 Discretionary Access Control Policies

Discretionary Access Control (DAC) is not a replacement of MAC, but it is to fine tune rigid constraints within MAC policy. The major difference between DAC and MAC is that DAC implements access control decisions of the owner of the object, while MAC implements decisions made by a central authority. In the scope of DAC, access control is granted to the creator of objects [21].

The creator or owner of the object specifies access permissions on that object and together with the attributes of users associated to the object, are stored in the authorization database. When any user requests access, the credentials presented are verified (authentication), and the reference monitor checks in the authorization database (Figure 2.4) for what the user is allowed to do or not. Access permission is then granted or denied according to access rights set by the owner based on the identities of users. Groups can also be defined in DAC by listing identities of members with special permissions to be included in the group.

DAC is flexible and is commonly used in commercial and government sectors [22]. For example, most commercial operating systems like UNIX and Windows use DAC as their main access control mechanism. Permissions may be granted based on need-to-know criteria, whom do I' like, or other rules at owner's disposal. The controls leave managing operations such as read, write and execute on objects at the discretion of the owner. Users are capable of passing access permissions to any other user or subject. The biggest drawback of DAC is basing controls on identities of users other than processes. It is hard to restrict access on an object by a process initiated by an authorized user. For example, in Figure 2.5, if user Anna owns a file 'f_1', and grants Bob read access to f_1, Bob can copy the contents of this file to a file he owns 'f_2'. Bob is now free to grant Char access to read the contents of f_1 indirectly through his own file, f_2 without Anna's knowledge. This makes DAC vulnerable to security flaws.

Maintainability of Authorization Database

In DAC, an individual user or program acting on behalf of the user, is permitted to specify the types of access other users may have to his or her objects. It means that access rights have to be specified for every single object in the system. If an access control list (ACL) is used as a mechanism for implementation, then all objects must have ACLs.

Definition 2.4 *A privilege p_x is a pair (id, m) granted to a user on an object x by the owner through an access control id.*

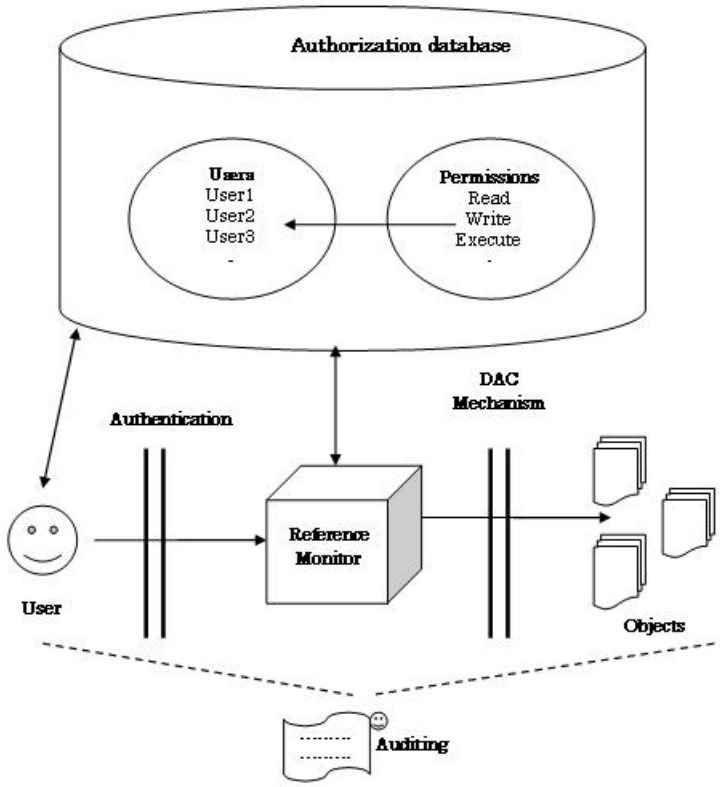

Figure 2.4: DAC Policy

Where, id is the user identifier or name and m is a set of access modes. For every object x created or added to the system, the owner has to update the database with an ACL entry of access rights given to users on the object.

Definition 2.5 *Let P_x be defined as a set of privileges on object x such that:* $P_x = \{(id_1, m_1), (id_2, m_2), ...(id_n, user_n)\}$*for individual users* 1 *to* n.

Suppose the database has many objects whose access rights need to be maintained. A universal set of privileges on all such objects has to be defined.

Definition 2.6 *Let UP be a universal set of privileges on all y objects in the database given by:*
$UP = \{P_1, ..., P_y\}$

As y increases UP also increases, hence an increase in authorizations to be maintained. In case of change in objects or subjects, the database has to be updated. It may seem simple and easy to review a set of users authorized to access a certain object just by retrieving the objects' entry, but still time-consuming. Suppose we are interested in establishing all privileges a specific user has in the entire system or we want to revoke some. Unless all the ACLs are searched, it is difficult to tell what privileges are granted to the user. Maintainability of authorization database becomes hard, since many authorizations need to be stored for all objects and the attributes of all users. This is not only administratively cumbersome but also prone to inconsistency and errors.

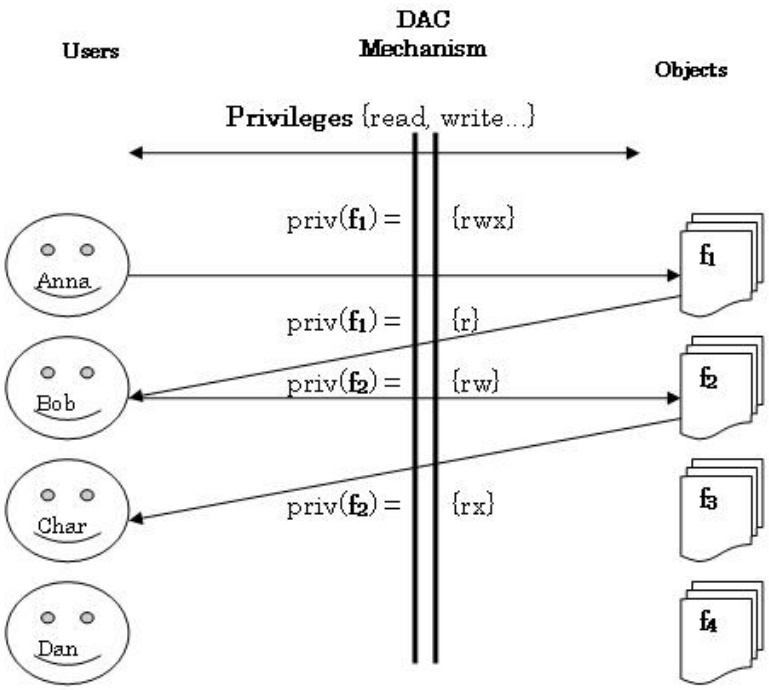

Figure 2.5: DAC Mechanism

Trojan Horse Problem

In DAC policies, users or any of the user's programs or processes can choose to share objects in their possession with other users. Moreover, ownership can be passed from a user to another user in discretionary access based on user or group identities, which makes DAC prone to Trojan horses [[25], [40], [43]]. DAC mechanisms trust users, but any program executed on behalf of the user, carry the identities of the user and hence the user's privileges on the object as in Definition 2.4. The program may now execute any access mode granted to the user with full authority. This seems to be okay with secure programs, but a Trojan horse violates the user's objective, and instead fulfills those of the initiator who planted it in the shared application or program.

Consider the example of DAC mechanisms earlier discussed in Section 2.2.3. If user Anna has her sensitive file f_1 and trusts only Bob among all users on the system to the extent that she grants him read access to f_1. Suppose another user Char is authorized to run shared programs with other users on the system, and has interest in Bob's file and is determined to access it. By inserting a Trojan horse in one of the programs, which can read and copy files from other users' files to his, Char would be able to get the contents of f_1 from whoever executes the file among the two without their knowledge.

Another example is if a user has 'grant' privilege with 'grant option' on an object, and he or she executes a program that has a Trojan horse embedded in it, the Trojan will also have ability to grant other users access to the same object without the user's knowledge.

2.2.4 Role-Based Access Control

Role based access control (RBAC) has recently gained popularity in use and research because of its flexibility. As formalized by Ferraiolo *et al* [13], RBAC introduces the role concept that captures permissions, subjects and objects. A complete discussion of RBAC is presented in Chapter 3.

2.3 Access Control Mechanisms

Access control mechanisms are used to implement access control policies. This section gives an introduction of common access control mechanisms in Section 2.3.1, and discusses access control matrix model and its associated mechanisms in Section 2.3.2.

2.3.1 Introduction

When securing a computer system, access control mechanisms provide a means for administrators to enforce security. Mechanisms such as user registration, privilege management, password management, access rights management may help in controlling access. Access control mechanisms act as guards to protected data and/or information and system resources. They are low level software and hardware functions which can be configured to provide efficient ways to implement an access control policy. Mechanisms require that security attributes about subjects and objects be kept for comparison in case of an access request. User attributes can be stored in terms of user identifiers, groups, roles, or/and clearance levels of trust on the user. Objects attributes can be stored in terms of sensitivity labels, types, and access control lists to mention a few. An access control matrix presents a variety of examples of access control mechanisms to enforce access control policies.

2.3.2 Access Control Matrix

An Access control matrix is the basic model for access control, though current access control systems do not implement it directly. It is described as a table in which, each row represents a subject, each column represents an object, and each entry is a set of access rights for that subject to that object [22]. It is the most known model for implementing DAC. Table 2.1 represents an access control

Table 2.1: Access Control Matrix

Subject	**Object**			
	f_1	f_2	f_3	f_4
Anna	read, write, execute, owner	write	read	read, execute, owner
Bob	read	read, write owner	write	
Char		read, execute	read, execute	
Dan	read, write		execute	read
Eddie			read	read, write

matrix for users Anna, Bob, and Char in relation with objects they access. Theoretically, an access control matrix is a good mechanism, but think of what would happen to a system with a very large number of users and objects. The matrix would grow large and become sparsely populated which makes it hard to use. In practice the matrix is instead represented in columns corresponding to access control lists, protection bits, and rows corresponding to capability lists as the most common mechanisms of implementation in computer systems today.

Access Control List

An Access control List (ACL) is a list associated with an object that specifies all the subjects that can access the object, along with their rights to the object [22]. An access list corresponds to a column of an access control matrix and its composition is a type tag which specifies the entry (for example user or group), qualifier field which describes a specific instance of the tag type, and a set of permissions that specifies access rights read (r), write (w), or execute (x) for the entry. An example of four access control lists extracted from Table 2.1 is presented in Table 2.2. Despite the ease to determine which users have access

Table 2.2: Four Access Control Lists

Object	**Access Control List**
f_1	User.Anna:rwx; User.Bob:r; User.Dan:rw
f_2	User.Bob:rw; User.Char:rx; User.Anna:w
f_3	User.Anna:rwx; User.Bob:r; User.Dan:rw
f_4	User.Anna:r; User.Bob:w; User.Char:rx User.Dan:x; User.Eddie:r

to an object, it is difficult to establish all privileges for a given user unless all the ACLs are searched. This is not only difficult but next to impossible in case of large systems with considerable user turnover. Hu *et al* [22] present another disadvantage with ACLs, that they are platform-dependent. Different systems have their own format of ACLs, which makes it hard for an organization considering changing from one policy to another. More so, when a requesting user is searched in the ACLs and access is denied or granted, the search stops without checking privileges of groups to which the user may belong. Despite these disadvantages, ACLs remain the best mechanism to implement DAC compared to other mechanisms discussed below.

Protection bits

The protection bit mechanism is related to the three mandatory entries in ACL but recognizes only one group, the owner's group. Owner is the owner of the object, group is a set of users to which the object owner belongs, and others are the rest of the world. Owner is allowed to set permissions on object in addition to the system administrator. The mechanism regulates file access rights (read, write or execute) in association with categories of users [22]. Table 2.3 shows access rights entries used in protection bits. Protection bits for files may include a type entry (- file, or d directory) such as:
$-rwx, rw-, r-$ or $drwx, r-, r-$

Table 2.3: Protection bits

Owner	**Group**	**Others**
Write, Read, Execute	Write, Read, Execute	Write, Read, Execute

An example of protection bits for files presented in Table 2.1 can be as follows:
$f_1 : rwx, rw-, r--$
$f_2 : rw-, r-x, -w-$
$f_3 : rwx, -w-, r-x$
$f_4 : rw-, rw-, r--$

A major drawback for protection bits is the difficulty of granting access to specific users that do not belong to the owner's group. There is only one group for each file, and no other way to classify other specific users. This would mean granting access to the whole world even when access is meant for an individual user or a specific group of users. With the growth of the Internet and computer networks, this increases exposure of information to attacks from everyone. Protection bits were commonly used in old versions of UNIX operating systems, but because of difficulties to specify privileges for an individual user or a specific group of users, newer systems include ACLs.

Capabilities

A capability can be thought of as a pair (object identity, access rights) on a protected object, with which a user is granted access to the object. The capability list is more attached to the subject, unlike ACL which is attached to an object. Examples of capability lists (extracted from Table 2.1) are presented in Tables 2.4 to 2.6. One major advantage of capability lists is their ability to show all accesses that are authorized for a specific subject. However, it is difficult to do the same with a specific object as it would require examining all capability lists, which makes their use less popular than ACLs in commercial systems.

Table 2.4: Capability List for Anna

Subject	**Capability**			
	Object:Operation			
Anna	f_1: read, write, execute, owner	f_2:write	f_3: read	f_4:read, execute, owner

Table 2.5: Capability List for Bob

Subject	**Capability**		
	Object:Operation		
Bob	f_1: read	f_2: read, write, owner	f_3: write

Table 2.6: Capability List for Char

Subject	**Capability**	
	Object:Operation	
Char	f_1: read, execute	f_2: read, execute

Profiles

Bertino *et al* [6] present an access control model based on user profiles containing both user interests and user credentials (a set of attributes characterizing users for access control purposes). Information is disseminated to users according to their interest profiles in such systems. Downs *et al* [11] describe the mechanism as that in which each subject is associated with a list of objects and access rights to those objects in the profile. The subject's profile is checked for required rights every time attempts to access an object are made. Downs *et al*, highlight problems associated with profiles as: restricting the size of the profiles, distributing access when an object is created or when access is changed, and determining all the subjects that have access to an object. Given the inconsistence in object names, the profile for a subject is liable to getting larger and difficult to manage. Also updating access to protected objects for different categories of users is difficult especially when changes occur or new objects are created. With such problems profiles are less used for implementing access control policies.

Passwords

The Password mechanism [[11], [25]] associates an object with a password that must be presented to the system whenever access is requested. It is one of the traditional DAC mechanisms used by operating systems. But currently it is more of an authentication than an access control mechanism. It is the password that the system checks to associate the subject with the object requested, and if there is a match access is granted, else it is denied. However, in case of an individual user having many objects to access, it requires the user to remember every password for each object. Storing passwords in files or programs is not secure since it is vulnerable to attacks. Passwords only work well if they are supplementing other access control mechanisms already in place.

2.4 Conclusion

This chapter has presented a definition of access control, summarized as a process of limiting use of objects to only authorized users by enforcing specified rules to control objects' use. It has also discussed traditional access control policies, MAC and DAC and presented examples of their implementation models such as MLS, BLP and access control matrix. In order to enforce access control policies, appropriate mechanisms are required, of which we have described the commonly used ones like ACLs, capability lists and protection bits. Despite the growth of MAC and DAC, security administration of large networked commercial systems is still complex. More so, these approaches have some deficiencies such as Trojan horse problems and rigid rules which are undesirable in most or-

ganizations. The evolution of Internet, large interconnected computer systems and large commercial and business enterprises requires flexible mechanisms for access control such as RBAC, which is discussed in Chapter 3.

Chapter 3

Role-Based Access Control

The origins of Role Based Access Control (RBAC) are related to the use of roles and groups in UNIX and other operating systems and database management systems [4], and separation of duty concepts [10]. The modern concept of RBAC that includes most of these ideas was first proposed by Ferraiolo and Kuhn [13]. RBAC emerged as a superior alternative to traditional mandatory and discretionary access controls [[35], [37]], which had some deficiencies that needed to be solved. Sandhu *et al* [38] describe a framework of models to understand RBAC and how it works. Because of it flexibility, RBAC received much attention, and was described in form of a security pattern by Yonder [48]. As its popularity and acceptance in the marketplace increased, there was a need to harmonize different models and descriptions. A standard model (NIST RBAC model) was proposed by Ferraiolo *et al* [15].

This chapter presents an overview of RBAC in Section 3.1 and discusses the components of RBAC Reference Model in Section 3.2. It presents RBAC as a security pattern in Section 3.3 and it is concluded in Section 3.4.

3.1 Overview of RBAC

Role based access control (RBAC) allows flexible controls over real world organizations' policies with minimal system abuse and errors. RBAC is policy neutral and is neither an extension of MAC nor of DAC, but tries to address the deficiencies of the two approaches. Some RBAC implementations can strongly simulate MAC [[28], [36]] while others can simulate DAC [39], depending on the purpose of the system. The essential concept of RBAC is where privileges are assigned centrally through roles which make it able to restrain users from having discretionary access to objects, and at the same time allow flexibility. Roles are relatively stable, while users and permissions may change rapidly. This flexibility allows its popular use in commercial enterprises.

In RBAC, permissions are granted to roles, and users are granted membership to roles based on their tasks or job responsibilities. This type of access control management is at a level that corresponds to the organization's structure and can help enforce organizational policies. The assignment of users to roles in traditional RBAC is done manually by the security administrator basing on the specified rules by the organization. A model extending RBAC to

automatically assign users to roles based on the attributes that users have, is presented in [2].

For the benefit of readers who are new to RBAC concepts, a clear difference between groups and roles need to be presented before we go any further. A group is a collection of users, but not a collection of permissions. A role is a mediator between two collections, a collection of users (group) on one side and a collection of permissions on the other side [38]. It can be simply referred to as a collection of permissions granted to a job function, a qualification or an expertise to perform an operation. A role exists as an entity separate from the role holder or administrator [33].

Most RBAC models are described in terms of users, subjects, roles, role hierarchies, operations, and protected objects [16]. But the basic elements for any RBAC system are those described in the core RBAC; users, roles, permissions (privileges), operations and objects. A user's process (subject) can access an object only if the user is authorized for a role. RBAC require that roles in the system are identified and users assigned to them, before they (users) or any program (subject) on behalf of the user can execute any operation on objects. In some RBAC models users are authorized to adopt different roles but in different sessions, while in others a user is allowed to exercise multiple roles at the same time [40]. A role can also be assumed by many users at once.

In figure 3.1, roles R_1 and R_2 are identified in an enterprise, users and permissions are assigned to them by the administrator. The services (authentication, authorization and RBAC) seem to be distinct, but in practice they almost happen together without much notice. When a user requests access to an object, the authentication service verifies the user for validity, the reference monitor compares user's credentials with the access authorizations stored in the database. This comparison is facilitated by authorization service. If the user is authorized for the role which has the right privileges for the operation, access is granted, otherwise it is denied.

An example of privileges on objects (files f_1, f_2, f_3 and f_4) enforced by RBAC is shown in Figure 3.2. Privileges are assigned to low level objects to make it simple, otherwise in real RBAC systems they are assigned to specific operations with meaning in the organization, such as 'make orders', 'credit account'.

Privileges on files, $priv(f_i)$ may be assigned to roles R_1 and R_2 as follows:

$priv(f_1) = ((R_1 : read, write, execute), (R_2 : read))$

$priv(f_2) = (R_1 : read, write)$

$priv(f_3) = (R_2 : read, write, execute)$

$priv(f_4) = (R_1 : read, write)$

When privileges are assigned to roles, users have to assume roles in order to perform operations on objects. From sets of privileges assigned to roles R_1 and R_2, there is nothing that changes when a new user joins the enterprise or an existing individual user leaves the enterprise. This is because privileges on objects are assigned to roles but not directly to users. Compared to what we have in DAC and MAC, this may reduce authorization rules that need to be stored. A discussion on authorization database is presented ahead in Section 3.3.1.

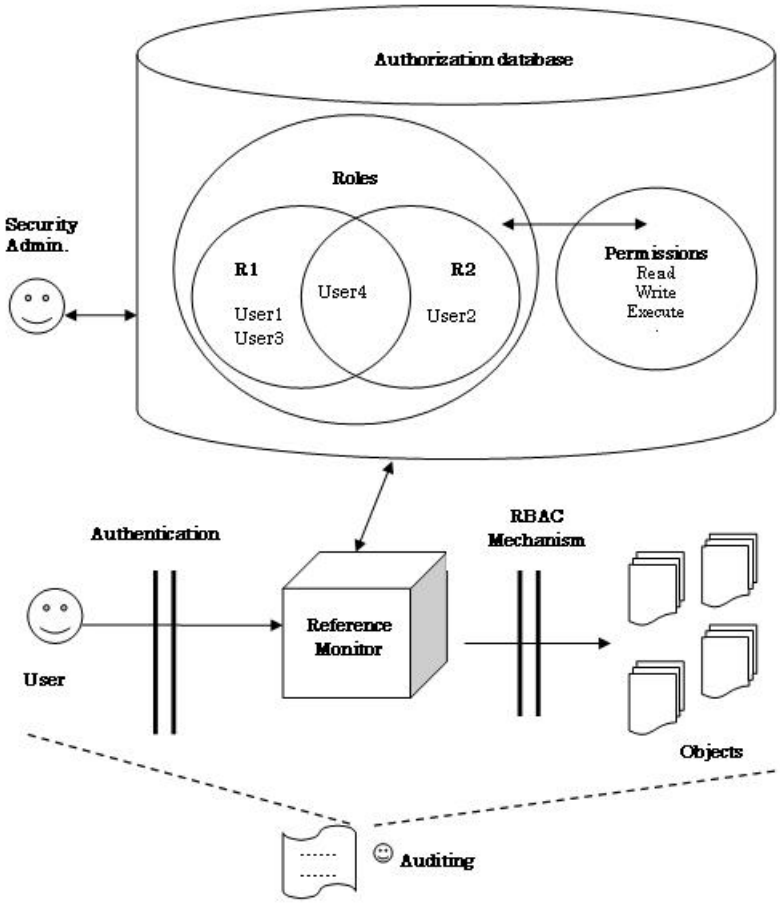

Figure 3.1: RBAC with other security services

3.2 Standard RBAC Model Components

The RBAC Reference Model [15] consists of four model components - core RBAC, hierarchical RBAC, static constrained RBAC and dynamic constrained RBAC. This section briefly discusses the four components and their functional requirements.

3.2.1 Core RBAC

Core RBAC is a prerequisite for any RBAC system, but the other model components (hierarchical, static and dynamic constrained RBAC) may be included depending on the purpose of the system. Core RBAC defines the following basic features that are found in all RBAC systems.

- *Users* - a user is defined as a human being for purposes of simplicity. But can include application processes, software agents, machines, networks and so on.
- *Roles* - a role is a job function within the context of an organization

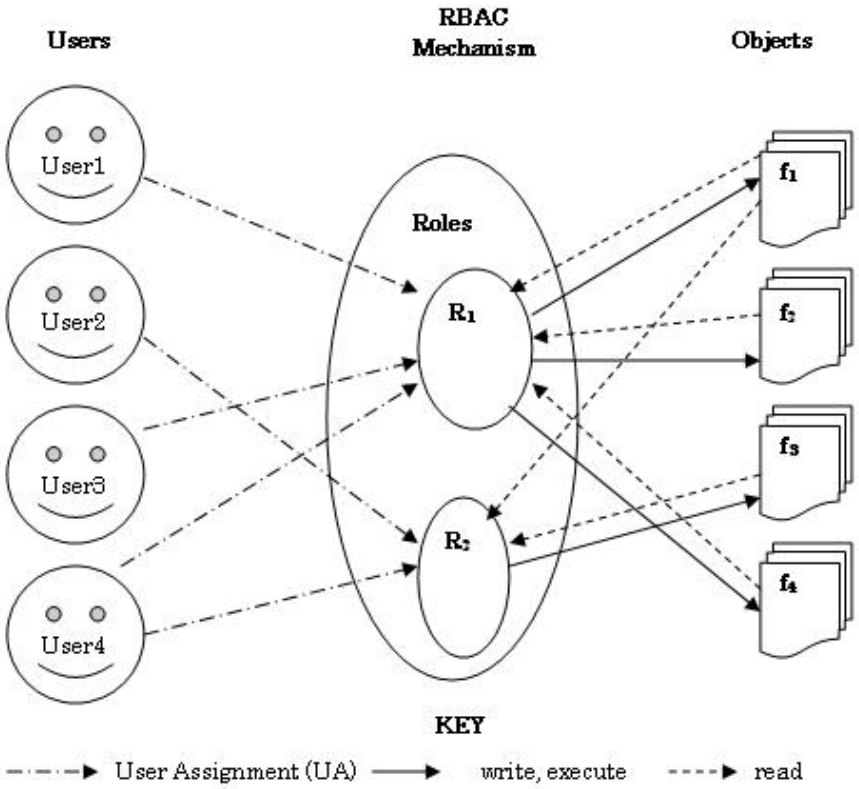

Figure 3.2: RBAC Mechanism

with some associated semantics regarding the authority and responsibility conferred on the user assigned role.

- *Permissions (PRMS)* - Permission is an approval to perform an operation on one or more RBAC protected objects. It can also be seen as a unit of access to information.
- *Operations (OPS)* - an operation is an executable image of a program, which upon invocation executes some function for the user.
- *Objects (OBS)* - an object is an entity that contains or receives information.

In Core RBAC, the principle is that user assignment and permission assignment can be many to many. A user can be assigned many roles and a role can have many users. Also a specific permission can be assigned many roles and a single role granted many permissions. User-role and permission-role relations are the basics of RBAC with a role as an intermediary. Roles are selectively activated and deactivated in a user session. A user can activate a subset of roles that he or she is a member of, and permissions from all activated roles will be at the user's disposal. This is in line with the principle of least privilege (need-to-

know privilege). If some of the privileges are to be restricted then constraints are added to the Core RBAC component as discussed in Section 3.2.3.

A user can also have multiple sessions with different permissions active at the same time, but a session is associated to a single user. Sandhu *et al* [38] equate a session to a subject in access control literature and define it as a unit of access control. Figure 3.3 (adopted from [15]) shows the association between the basic features of Core RBAC in this discussion. Double arrows show a many-to-many relationship and single arrows indicate a one-to-many relationship.

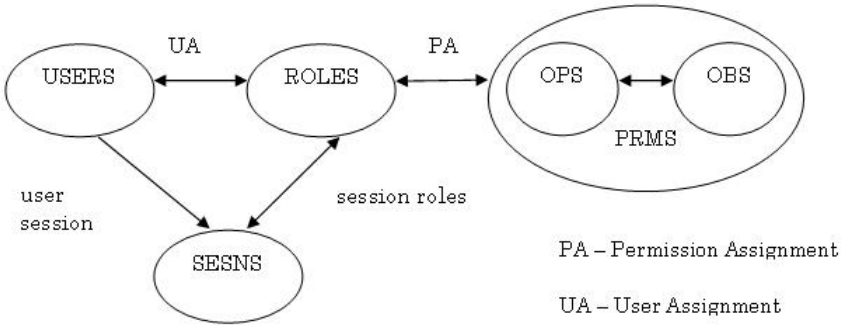

Figure 3.3: Core RBAC

Functional Requirements for Core RBAC

Ferraiolo *et al* [15] highlight RBAC system and administrative functional specifications for standard RBAC model. Elements and relations for Core RBAC require functions to create, review, manage and maintain them. These are administrative, supportive and review functions that can be embedded in the implementation to help in meeting such requirements. The implementation and actual commands may vary from system to system but the functional intention remains the same.

Administrators create and delete users and roles using *AddUser()*, *AddRole()* and *DeleteUser()*, *DeleteRole()* functions respectively. They also institute relationships between roles and operations on objects by assigning and removing users for user-role relationship, and granting and revoking permissions for the permission-role relationship using *AddUser()*AssignUser(), *RemoveUser()* and *GrantPermission()*, *RevokePermission()* functions respectively.

To support and manage users and roles, supporting functions are also required for session management and accesses control decisions. These include functions for creating and deleting sessions (*CreateSession()* and *DeleteSession()*), adding and dropping active roles (*AddActiveRole()* and *DropActiveRole()*) and checking access (*CheckAccess()*). A new session is created for a given user as long as the user exists in the system as a member of that active role. It is against the same principle that a given session is deleted if it is a member of sessions owned by the given user.

A role is added as an active role of a session of a given user if and only if the user exists in the system as a member of the role and the current session is owned by that user. The function for dropping an active role works along the same

lines as long as the role is an active role of the rightful user's current session. The function for checking access (*CheckAccess()*) is to ascertain whether a user or a user's process of a given session is allowed to perform the operation on a specific object. Access may be allowed only if the session is owned by the rightful user, the object is among a set the active role is authorized to access and the operation is among those authorized for the active role.

Review functions are required in Core RBAC to be able to review the users assigned to a role and vice versa, these may be in form of *AssignedUsers()* and *AssignedRoles()*. Reviews for permissions of a valid user through assigned roles and active roles associated with a valid session, operations a given valid role is authorized to perform on a valid object, and operations a given valid user in a valid role is authorized to perform on a valid object and many other reviews may also be required in a particular system.

3.2.2 Hierarchical RBAC

Hierarchical RBAC adds the role hierarchy to Core RBAC to cater for increased numbers of roles. In large organizations, the number of roles can be extremely large and a system becomes inefficient, then, the whole idea of roles would have lost meaning. Some job functions (roles) in organizations overlap with each other, and at times users have similar sets of permissions on some objects. In such situations, administrators are allowed to define roles with respect to other roles with which they relate in terms of capabilities. The overall set of capability relationship is what is referred to as a role hierarchy [24].

To understand the role hierarchy concept, let us use an example of a research group like Security of Systems (SoS) group at Radboud University Nijmegen. Suppose its membership is composed of teachers and students and have some files accessible by all its members through role 'Member', and other files' access limited to sub-groups according to their tasks (studying, teaching, administration and so on). Suppose in the group there are some students who help with teaching responsibilities, such as PhD students. A role of *Student Teacher* may be defined to contain other roles like *Teacher* and *Student*.

Role hierarchy uses an inheritance relation. In addition to other privileges assigned to role *Student Teacher*, it inherits privileges from *Teacher* and *Student* roles. However, it may be necessary to limit the scope of inheritance for security reasons or any other justifiable reason. There might be information students want to keep to their role for some reasons. May be they are working on a project whose results are not yet clear and would not want student teachers to know about it. The only way would be to keep some privileges private by defining another role, say *Student_1*.

On the other hand some teachers may belong to a senior role with other administrative functions that student teachers need not to know. Role *Lecturer* may be defined to prevent inheritance of some privileges by student teachers, but *Lecturer* can fully inherit all privileges from *Teacher*.
Figure 3.4 shows an example of role hierarchy tree that results from this discussion.

Role hierarchies provide a means of reflecting authority, competence and responsibility in an organization. Inheritance can be for both role privileges and user membership. In Figure 3.4, role privilege inheritance is bottom-up, while user membership inheritance is top-down.

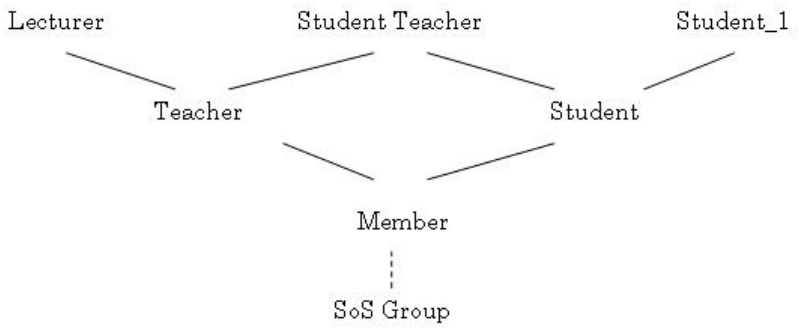

Figure 3.4: Simple example of Role Hierarchy

Functional Requirements for Hierarchical RBAC

Since Core RBAC is required in any RBAC system, all its functional specifications are embedded in the implementation of Hierarchical RBAC. Additional functions are added or some are redefined to cater for role hierarchies. These include administrative functions for adding and deleting an inheritance between two existing roles, and creating new roles and adding them as ascendants or descendants of existing roles.

Support system functions to create a session and to add an active role in Core RBAC functions are redefined to include or activate inherited roles in a new session. Likewise, additional functions for reviewing authorized users directly assigned to a given role and members of roles that inherit the role in question, authorized roles directly assigned to a given user and roles the user inherits through the directly assigned roles are defined. A reader of this thesis, interested in the details of these functions is encouraged to read Ferraiolo *et al* [15].

3.2.3 Constrained RBAC

Sandhu *et al* [38], equate constraints to negative permissions to restrict an authorized user. The principle of least privilege in Core RBAC is used for a user to have permissions necessary for one or more sessions to perform required tasks and no more. To enforce such constraints, Separation of Duty (SOD) principle is applied. Access is permitted to objects that are related to the duty to be performed. The importance of SOD to enforce conflict of interest policies, started as early as the 1980's in government and commercial sector. Clark and Wilson [10] use the SOD principle in their model to prevent fraud and other errors. To control users, constraints are applied at the time of access or while assigning users to roles. This is because users need to be controlled not to misuse their job positions in the organization, but also have enough permission to perform their duties.

SOD relations can be in form of static separation of duty (SSOD) or dynamic separation of duty (DSOD). Adaikkalavan and Chakravarthy [1] give other extensions of constrained RBAC to support context-aware constraints, content-based constraints, control-flow dependency constraints, and so on, to

make it more powerful, but these are beyond the scope of this research.

Static Separation of Duty

There are many ways of enforcing SOD policies such as restricting user-role assignment over two conflicting roles. In the research group example discussed in Section 3.2.2, Static Separation of Duty (SSOD) constraints can be applied to user-assignment for roles *Student* and *Lecturer* since they may cause a conflict of interest, if assigned together. Some files may contain examinations for students, which would lead to exam leakages if role *Student* is granted all privileges of role *Lecturer*. Therefore a user who is a student cannot be authorized for both roles in the group since there is a strong conflict of interest. SSOD relations define constraints that are applied to a user's entire permission space in constrained situations, and in this particular case a user of the system who is a student.

Dynamic Separation of Duty

Another way to limit permissions that can be granted to a user is by applying Dynamic Separation of Duty (DSOD) constraints, say at the time of accessing objects. While assigning roles to users, some constraints may be applied so that operations to be performed by role members are controlled. The essence of DSOD is to define constraints which limit the availability of permissions on roles that can be activated in a user's session at any given time. DSOD defines constraints which are applied such that a user is allowed to assume two or more roles which do not cause security concerns if activated independently, but may not be activated together to create a conflict of interest. A user may be a member of more than one role, but to prevent fraud and other related conflict of interest, only one role can be activated at a time but not both roles.

For example a user who is a *Cashier* may have another role of a *Cashier Supervisor*, but may not activate both roles simultaneously since they conflict. The user can only be authorized to activate the roles independently. Likewise the same user may be a customer with an account in the same bank. The policy must define constraints which restrain the user who is both a *Cashier* and a *Customer* from abusing his or her position, hence may not process his or her personal account while processing accounts of others.

DSOD can also be applied in form of limiting the number of users that can assume a role. For example, the role of a project coordinator may be constrained in such a way that it can only be assumed by only one person at any given time. If there is one user already authorized for that role, no any other user can be granted access to the same role, unless the former is not on duty and the later is authorized to deputize.

Functional Requirements for Constrained RBAC

RBAC systems can apply constraints with or without role hierarchies. But in Hierarchical RBAC the SOD principle is enforced by defining role hierarchies including inheritance of SOD constraints. When a user requests for a role, the system checks the role for SSOD and DSOD relationships before granting access to the requested role. Roles with SSOD relationships may never be entrusted with the same individual user even in Hierarchical RBAC. The same applies to

roles with DSOD relationships; they will never be activated simultaneously by an individual user at once.

Functions to confirm and guarantee that a given user-assignment relation does not contravene constraints associated with SSOD, and at the time of role activation in a given user's session, the constraints associated with DSOD are applied, are well explained in [15]. The depth of implementation depends on the use and model of the system.

3.3 Authorization Database and Other Issues

RBAC has key services that make it popular as compared to MAC and DAC. These include its ability to make the maintainability of authorization database easy, and the capacity to limit like Trojan horse problems, both discussed in this section.

3.3.1 Maintainability of Authorization Database

Though privileges may be given directly to users in RBAC, the number of users is minimized by use of roles. Users get privileges through assuming roles. Adaikkalavan and Chakravarthy [1], say that enterprises are trying to exploit the use of RBAC to reduce the cost for authorization management. In RBAC, authorizations take two forms; users are authorized for assigned roles, and roles are authorized for assigned privileges, which simplify their management.

Definition 3.1 *A privilege, p is a pair (O, m) granted by the administrator to a role r through an access control 'rid'.*

Where,
O is a unique name or identifier of an object
m is a set of operations to be performed on the object by the role, for example, *credit* and *debit* operations on *an account* by a role *Cashier*
'rid' is a unique name or identifier of a role.

Definition 3.2 *For a specific object x, let a set of privileges p_x be defined for a specific role r_1 as:*
$p_x = \{x, m_{xr_1}\}$

Definition 3.3 *Let authorizations between role r_1 and the object x be defined as:*
$(auth)r_1 = \{rid, p_x\}$
For role r_1 to access n objects, the total number of authorizations stored will be:
$(auth)r_1 = \{rid, (p_1.....p_n)\}$

If authorizations were to be stored for individual users, then the database would be very big, with authorizations for all individuals required. But with roles, when a user changes tasks because of, say, changing a department, or is promoted, the user's current role(s) which are not required by the new position may be removed, and new role(s) assigned. This will not affect the privileges for operations assigned to any of the roles (authorizations between roles and objects will remain intact). But if privileges are directly assigned to users, then

all existing privileges have to change and new ones assigned when users' tasks change. This would be a complex task which is simplified by having authorizations between roles and objects instead of users and objects.

On the other hand, objects can also be many in an organization, such that management of authorizations between individual objects and roles becomes complex. Sandhu and Samarati [40] say that objects can be classified according to their types (such as application areas) in RBAC. This makes authorization administration easier and well controlled. Access authorizations are between object classes and related roles but not individual objects. Instead of specifying authorizations between roles and each specific object, object classes are used. Furthermore, any new object created will automatically take on authorization specified for the parent class, hence no need for new authorizations. For example in a research group objects like documents can be classified according to their application area, say, lecture notes, research papers, administrative files, and so on. This would mean that *Student* role can be authorized to read the whole class of lecture notes instead of a single file, and *Lecturer* role can read and write the entire class.

Instead of $(auth)r_1$ in Definition 3.3, having many attributes of $\{p_1, ..., p_n\}$ to be stored, objects 1 to n may be put in one class C with a unique class name or identifier c, and a set of privileges PC defined on class C.

Definition 3.4 *A set of authorizations between role r_1 and class C would be defined as:*
$(Auths)r_1 = \{rid, PC\}$

Where, PC is a pair (c, m), and m as in Definition 3.1, but on the whole class C instead of an individual object. Any new object created would not require independent authorizations if it falls under an existing class.

3.3.2 Ability to limit Trojan horse problem

The principle of least privilege is an important reality that when implemented in any RBAC system it addresses the problem of Trojan horse. Users are prevented from unintentional error, by steering them to only those parts of the system required for their tasks for efficient performance. A user cannot pass privileges to another user, and a role cannot share assigned privileges with another role unless they are related in a role hierarchy by inheritance. Fernandez and Pernul [12], present a pattern for session-based role based access control and emphasize the effect of an access session on Trojan horses. A session makes it harder for an attacker to have all privileges for a role apart from those activated for the current user's session. More over, the session may even expire before the attacker succeeds in getting all the privileges required. It is also easy to exclude roles that may violate the policy when a session is opened by applying DSOD. But what if the activated privileges in the current session are enough for the attack in the basic RBAC model?

Chou proposes a multileveled RBAC in [9] as an extension of RBAC96 with features that ensures secure information flow and avoids Trojan horses. These features in addition to the principle of least privilege, separation of duty, the concept of roles and the use of sessions, make Trojan horse vulnerability less possible in RBAC.

3.4 RBAC as a Security Pattern

RBAC is discussed along many lines, and it has been presented as a security pattern [[42] [48]] to act as a reusable solution for recurring security problems in systems. This section describes the RBAC security pattern similar to that presented by Schumacher*et al* [42].

Description of RBAC security pattern: This pattern describes how to assign rights based on the tasks people perform in an environment in which controlled access to information and system resources is required. It describes how users are assigned rights based on their job functions or assigned tasks despite large numbers of people, information types, and a variety of resources that may exist in an organization.

Pattern name: RBAC pattern

Example: A financial institution has many employees at the headquarters and in its branches in different locations. It also deals with other financial institutions, government institutions, vendors and other customers. It has different information types and a variety of resources whose access must be controlled. Defining individual access rights has become a time-consuming activity and has increased security administrative costs. At the same time it is prone to errors and may lead to loss of information or compromised security. Due to the nature of information handled by the institution, separation of duty would be desirable to avoid an individual controlling a process from start to finish.

Context: A variety of job functions or tasks in a financial institution require different skills and responsibilities. Users can be classified according to their job functions or tasks and should be assigned rights to access information and resources according to the needs of their job functions. Job functions correspond to roles played by individuals in performing their duties.

Problem: There should be a centralized, flexible and efficient way of assigning rights to users other than individual rights which require storing many authorization rules. How do we assign rights to users according to their functions or tasks in an organization?

Forces to be resolved:

- Individual users in an organization have different needs for access to information.
- Misuse of Information and Resources could disrupt institutional operations and cause financial, legal, personal privacy, human safety and other related impact.
- Users can be classified according to their tasks, and common tasks require similar sets of rights.
- Granting rights to individual users would require storing many authorization rules and this would also be hard for administrators to keep track of these rules.

- Users may also perform more than one role and other policies such as separation of duty may be enforced.
- A role may be assigned to individual users or to group of users.
- A user may have more than one role depending on his or her tasks.
- The organization needs to define access rights for its people according to a need-to-know policy.

Solution: Registered users in the institution are assigned roles based on their job functions. Roles are granted privileges according to the need-to-know policy. Privileges associated to a role define the access types the user with in a role is authorized to apply to the protection object. The solution is based on the classic model in Figure 3.5, adopted from [42] which is an extension of authorization pattern.

Implementation: Making roles correspond to tasks instead of job titles is the best approach in an institution with specific job titles. Job titles may confuse though they can be used in some organizations. For example a job title of a Principle Banking Officer (PBO) in a financial institution may be given to officers at that rank but with different tasks. This would confuse if a role of PBO is given to an individual who has tasks of system administration as well as that with accounting tasks (system administrator and an accountant). It is therefore fit to use tasks since they usually correspond to the actual functions performed by individuals in an organization. In this case we would have a role 'Sysadmin' as a collection of system administration tasks and that of 'Accountant' for accounting tasks.

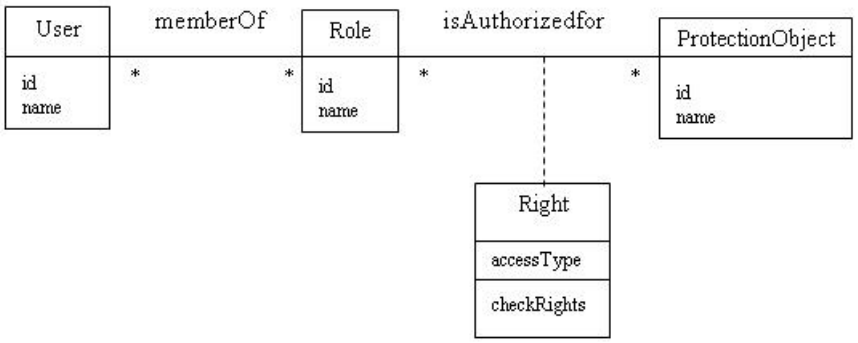

Figure 3.5: RBAC Model

Consequences: A variety of benefits is expected from implementing RBAC pattern.

- Organization policies about job functions can be reflected directly in the definition of roles and the assignment of users to roles.
- Administrative effort is reduced because rights are assigned to roles instead of users. In any case there are many users than roles in an organization.

- Roles can be structured for further flexibility and reduction of rules into role hierarchies.
- The implementation of least privilege principle is supported because no role is granted more rights than required.
- Administrators can use RBAC mechanisms to enforce a policy of separation of duties, a valuable policy in deterring fraud.
- Accommodating users arriving, leaving or changing job functions becomes easy in case of big turn-up. These actions are only manipulation of associations between users and roles.
- Groups of users can be used as role members, hence reducing the number of authorization rules and the number of role assignments.
- Classes of objects can be used instead of individual objects.
- Above all, increased efficiency, low maintenance and reduced security administration costs, are registered when RBAC is well implemented.

Possible liabilities include:

- An extra layer of complexity is added for developers due to the new concept of roles, assignments to multiple roles and so on.

Known Uses: RBAC is implemented in a variety of commercial systems, including Windows OS, Oracle Database Management Systems, Sun's Solaris OS and a standardized model by NIST.

Related patterns:

- Authorization pattern [42] describes who is authorized to access specific resources in a system whose access is controlled.
- Reference monitor [42] enforces declared access restrictions when an active entity requests resources.
- Roles rights definition [42] provides a precise way of assigning rights to roles to implement a least-privilege policy.
- Pattern language for implementing RBAC appeared in [27]
- Session pattern [34] describes a way of protecting resources from misuse because during a particular session a subject can only access resources allowed by the activated authorizations.

3.5 Conclusion

Role-based access control (RBAC) is a newer approach to access control using the concept of roles to grant privileges on objects. This Chapter has discussed the basics of RBAC, and with examples described how it works. Its relationship with other related security services (authentication and authorization) and how privileges are assigned to users through roles have been presented. We have also

discussed the major components of the RBAC Reference Model; Core, Hierarchical, and Constrained RBAC (Static Separation of Duty (SSOD) and Dynamic Separation of Duty (DSOD)) as described by Ferraiolo *et al* [15]. The chapter has also presented RBAC as a security pattern embedded in software development and its known uses in Operating Systems and Database Management Systems as described by Schumacher*et al* [42]. This chapter gives an insight of how RBAC resolves issues that were left hanging by MAC and DAC, and acts as pillar for RBAC implementation in the case study.

Chapter 4

Case Study

4.1 Introduction

Common access control approaches (MAC, DAC and RBAC) that have attained tremendous success in providing security to information systems were discussed in previous chapters. Nevertheless, some organizations in developing countries may not even have any of these in place, yet deal with critical information. This means that ad hoc access control is still used by many organizations in these countries. More so, the level of research in information security, and related technologies is higher in developed countries than in developing countries. A different approach is taken in this research to understand security of systems based on a case study in a developing country - Uganda.

While different access control approaches may have different capabilities, in most cases, an organization is the sole owner of information and users act on its behalf in capacities entrusted to them. This means that access control decisions should be based on the specified duties, responsibilities, and qualifications required by the job. It is against this back ground that RBAC has gained momentum to permit access to information basing on roles individuals play in an organization. This case study seeks to understand the extent the older approaches (MAC and DAC) are used in developing countries as well as the drive to adopt newer approaches like RBAC, and how it is utilized.

This chapter presents the motivation, general and specific problems, and discusses the findings of the case study and the recommendations.

4.1.1 Motivation

From literature study, it was realized that information security mechanisms in general and access control in particular is evaluated basing on surveys and implementations done in the developed world. Our inspiration is that, there must be issues that need to be understood and resolved, which hinder adoption, development and deployment of these technologies in the developing world. Current research is focused on expanding and developing new technologies based on the old ones. But are the old ones well understood and utilized in all countries? Will the new technologies be appreciated when the old ones are not yet understood and utilized? How do we help the developing world to participate fully

in this information technology era? To find out facts and help future research, the following were among the questions the case study seeks to answer.

- What are the access control policies, technologies and mechanisms in place and what influences them?
- What are the methods of work and implementations of existing technologies in local organizations in developing countries?
- What are the benefits and limitations of these technologies?
- What are the future plans as far as these technologies are concerned?

4.1.2 General Problem

As organizations grow, the number of employees and Information Systems also increases to meet different organizational requirements. This in turn increases information amounts to be processed, stored and disseminated as well as the risks of unauthorized access. An organization must decide on who accesses what and when according to its information security policy.

More so, most researchers in the developed world usually define, develop and present their findings using complex methods that are hard to understand by users in developing countries. On the other hand, some users just buy software and hard ware products without understanding the underlying technologies. As a result, critical information has continued to suffer at the hands of both authorized and unauthorized users, even when heavy investments in related technologies and mechanisms have been made.

4.1.3 Specific Problem

A financial institution in Uganda, which is at an International level, has critical information systems and a large number of employees. By the nature of business conducted by the institution, it deals with other financial institutions within the country and other parts of the world. It has other customers, suppliers and vendors to deal with. It also has branches up country, which may be connected into a Wide Area Network (WAN) in the near future.

To cope with the required level of information processing, storage and dissemination as well as system maintenance, its Management Information Systems (MIS) department has registered an increase in staff numbers, as well as acquiring new database management systems and operating systems (OS). MIS staff has to access information systems and computing resources to perform their duties. Given their level of participation in information systems, they are likely to abuse systems accidentally or intentionally, unless their actions are controlled. The study therefore, focused on MIS department to understand how systems and technologies in place are utilized.

4.2 What the Status Was

At the inception of the study, the few documents available were studied; these included the general security policy in place, organizational structure and some standard operating procedures for implementations in place. Since no RBAC

system is implemented by the institution, we had all the privileges given to their employees to access the LAN, which enabled us to gain access to few more documents available on the network. Our attempt to use questionnaires was unsuccessful since they could not be filled and returned in time. Nevertheless, the best tool at our disposal was observation and participation in activities which yielded fruits. Some informal interviews were also conducted with staff during casual interactions.

During the study we observed that, while security is no longer a private concern but a threat to many, some organizations still find it hard to discuss how to improve security of their systems. Instead of strategically identifying security flaws to practically counteract them before they cause loses, these organizations tend to rely more on deployment of physical protective measures. This method of work does not only hinder building and developing secure systems, but also deceive the owners that all is well without testing, analyzing and evaluating their systems internally.

Despite, the number of critical information systems present in the Institution, there is no clear access control policy in place to state clearly how information should be accessed. However, we observed an ad hoc access control mechanism in use, with bits of common approaches applied as to when administrators deem it fit. User accounts for all employees are created and maintained by the MIS department and privileges to individual users are assigned as to when they are required without any formal policy. This means storing many authorization rules as well as revoking or granting permissions for each member that comes or leaves the Institution. This is not only administratively cumbersome and costly, but also prone to errors.

In the MIS department, some functions are performed by groups of users using common usernames and passwords. Auditing such actions becomes a problem since tracking the action to an individual user is not easy. A simple example is the use of 'local_admin' as a user name with a common password by all staff in Operations Section to perform their user and technical support tasks that requires admnistratative privileges. Another example is a common username 'root' with a common password for all root users to perform their tasks. These are only a few examples but most of the system accounts had the same problem, including the 'oracle' user for the oracle database. We appreciate the fact that there might be technical reasons for doing so, but our argument is that, individuals should first log on to the system with their user identifiers (UIDs) before assuming system accounts.

Since the Institution has already invested in new systems (Solaris OS and Oracle databases) with features capable of handling some security issues such as eliminating anonymous logins, we were interested in finding out why the features are not well implemented and configured. Among the new acquired OS, is Solaris 10 known for trusted computing environment with its RBAC features. Solaris 10 is not yet migrated to any system server but under test. Solaris 9 is hosted on nine servers and other three servers have Windows OS. However, there are many factors which may influence the way security issues are handled in such organizations. We observed the following:

Lack of Appropriate knowledge: While some users are willing to adopt new technologies and mechanisms, they have little knowledge of how they work

and the benefits to the organization and individual tasks. Necessary information is either not available to users or what is available is too complex for a novice user to understand, or users are not provided with enough training and explanations.

Lack of expertise: While new technologies might be acquired because of their capabilities read in publications, advertisements, and marketing strategies of the vendors, some organizations lack the man power to utilize the technologies for maximum benefits. As a result a small percentage of the technology capabilities is utilized after heavy investment in its purchase.

Lack of enforcement controls in software: While customers may lack appropriate knowledge and expertise, systems are also developed without much emphasis on alerting customers to configure embedded security features during installation and operational time. Configuration of most features is left as optional to customers. If customers are ignorant about such features they are left unutilized. System developers may need to think of additional means to alert customers of the presence of security features to ensure good security practices.

Limited availability of Internet: While Internet is the most known source of current information, its availability and limited bandwidth is still a big problem in Uganda. Administrators get limited help from the Internet to enhance their knowledge.

Much Trust in Physical Security: A lot of investment is put in physical access control which is a good step to protect information, but this may mislead the company on other access control mechanisms required.

Meaning of Access Control: Access control to some organizations may be limited to authentication and authorization. As long as a user is authenticated and authorized to use a resource, no measures are in place to control the way the user utilizes the resource. Auditing log files, which appear to be a means of monitoring information resources in the Institution, cannot replace actual access control mechanisms and does not help much where accounts are shared by many users who login directly.

Organizational culture: People in organizations are shaped by the organizational structure and culture. Unless the old cultures in such organizations change, Information Technology (IT) with its dynamic nature may take long to develop. Top management of such organizations need to support new innovations, and let people with skills develop them, however, the bureaucratic nature of these institutions often frustrate upcoming IT ideas.

Rate at which IT professionals come and leave the Institution: The MIS department registered the highest number of employees that left the institution during the period of the study. Many of them were among those trained to handle new systems. Many reasons such as; lack of clear policy on promotions and salary increments, lack of professional support and so on, may cause this,.

Constrained budgets: While some technologies may be purchased, the budgets may not cater for their deployment and development or even training staff. The technology may even require hiring some experts for some time, as the company staff learns to work with it, but if the budget does not cater for this, then the technology will become obsolete before it is utilized.

Limited financial resources: Some security measures may be hard and costly to implement, hence not considered a priority given a long list of issues to be solved.

Corruption: The system may be intentionally left with some security vulnerabilities by individuals interested in taking advantage of its security relaxity.

4.3 RBAC Implementation

Since it was clear that there is no access control policy in place, it was not feasible to invest in a new access control system. However, the MIS department was convinced that the Solaris 9 Operating System and the Oracle Database Management Systems in place were not fully utilized as far as their RBAC features are concerned. This section presents the planning for implementing RBAC at organizational level and the summary of our achievements in having the Solaris 9 RBAC features utilized.

4.3.1 Planning for Organizational RBAC

Setting up an RBAC system is not a simple task. A master plan has to be in place if maximum security and business value is to be extracted from RBAC.

1. Information on Systems, Hardware and Software is compiled by identifying servers, databases and applications to determine their level of security required basing on the core mission of the organization.

2. With the help of the Human Resource (HR) department, a comprehensive list of job functions is compiled and together with relevant managers, roles are determined. This helps in categorizing roles and determining role-based access rules.

3. Detailed plans of how roles will be changed, updated and how users will be assigned to these roles, including creating and terminating accounts in a timely manner is thought of. These plans should be approved by the organization's top management since information and information systems belong to the organization.

4. The IT team is responsible for transferring each application's embedded security functions into a new centralized system, including home-made applications and customized commercial applications before the system goes live.

5. It is also important to educate and train organizational staff using a top down strategy for rapid acceptance of RBAC. One thing management has to understand is that while RBAC gives full control of information to

the organization, it reduces the powers of control of users on objects they create. But if employees clearly understand how and why RBAC is vital to the organization's information security and realize how it can make them more productive, they are likely to embrace the system without hesitation.

We used the Solaris 9 Operating System (OS) RBAC features and its implementation to configure some roles according to tasks performed by individual users in the MIS department.

4.3.2 Solaris 9 Operating Environment RBAC

The Sun Solaris Operating System, like any other UNIX-like OS use DAC as its primary access control mechanism for files. However, it provides environments that embrace RBAC features to aid their customers adopt better security practices. The Solaris 9 Operating Environment (OE) has an improved implementation of RBAC than Solaris 8. Solaris RBAC provides an alternative to the standard UNIX model with an all powerful superuser called root [[8], [30], [32]]. Instead of having a superuser doing everything on the system, special user accounts in form of roles are assigned to individual users according to their job functions. When a role is established to handle a set of job functions, then those functions are removed from root's capabilities. This limits the ability of an individual who has root password to damage the entire system.

Solaris RBAC provides a means of reallocating system controls, but it is the organization's responsibility to decide the degree of implementation according to its security needs. It enables a variety of security policies through the flexibility of roles. Solaris 9 RBAC is based on users logging in to the system using their real user identifiers (UIDs) and assuming special roles that enable them to run restricted tools and utilities. Auditing becomes easy as the actions of a role are traceable to the user who assumed the role. RBAC introduces the following features into the Solaris OE.

- *Privileged Application* - An application that can override system controls and check for specific user UIDs and group identifiers (GIDs), or authorizations.
- *Role* - A special identity for running privileged applications that can be assumed by allowed users only
- *Authorization* - A permission that can be assigned to a role or user (with in a rights profile) to perform a set of tasks otherwise not permitted by access control policy.
- *Rights Profile* - A package that may consist of authorizations, commands, and additional rights that can be assigned to a role or user.

Rights profiles are assigned to roles and roles are assigned to users to enable them to assume roles whenever their tasks require special capabilities. Direct assignment of authorizations and rights profiles to users is possible but not practically secure, hence not advised. Solaris RBAC roles are created in the same way as user accounts, with home directory, group, and password and so on. Information on a role is stored in related databases such as *passwd*, *shadow*, *user_attr*, and *audit_user* databases.

Users cannot log in directly to a role, but must login to their user accounts first, and cannot assume a role directly from another role. A user can assume only one role at a time. This satisfies the least privilege and the separation of duty principles. Solaris RBAC is flat for roles; when a user assumes a role, the role's attributes replace all the user's attributes. However, the user's real UID is important for auditing a role's action on top of the role's ID. Authorizations and rights profiles, may be made hierarchical through wildcards (*) and the ability to assign supplementary profiles to other profiles respectively.

No configured roles are shipped with the Solaris 9 software. Roles are set up for special tasks as the organization deems it fit. However, there are three roles recommended for the Solaris 9 RBAC, though it is not a requirement that they are implemented. They can be configured by assigning the appropriate predefined rights profile in the RBAC software. The recommended roles are:

- *Primary Administrator* - Very powerful and equivalent to root. It can perform all administrative tasks, and can grant rights to others.
- *System Administrator* - A less strong role for administration that is not related to security. Can add new users but cannot set passwords or grant rights to other users.
- *Operator* - A junior administrator role for operations such as backups and restores, and printer management.

Data for RBAC elements is stored in the following databases.

1. *user_attr* - this is an extended user attributes database that associates users and roles with authorizations and rights profiles.
2. *auth_attr* - an authorization attributes database that defines authorizations and their attributes, and identifies the associated help file.
3. *prof_attr* - a rights profile attributes database that defines profiles, lists the profile's assigned authorizations, and identifies the associated help file.
4. *exec_attr* - a profile execution attributes database that identifies the commands with security attributes assigned to specific rights profiles.

These databases are fully described in [31], but we give a brief summary of their contents in order to prepare the reader for our discussion in Section 4.4.

The *user_attr* database stores extended user attributes (such as authorizations, rights profiles and assigned roles). Information in the *user_attr* database complements that in the *passwd* and the *shadow* databases. A user and a role can be differentiated by the type field. Type fields in *user_attr* database are as follows: *user; qualifier; res1; res2; attr*
Where,
user - the name of the user or role as specified in the *passwd* database
qualifier - Reserved for future use
res1 - Reserved for future use
res2 - Reserved for future use
attr - An option list of key-value pairs (type, auths, profiles, and roles) separated by semicolon which describes the security attributes to be applied when the user runs commands.

The *auth_attr* database stores all the authorizations that are to be assigned to rights profiles, but they can also be assigned directly to users though not advised. The fields are as below:
authname; res1; res2; short_desc; long_desc; attr
authname - A unique character string that is used to identify the authorization prefix [suffix]. For the Solaris Operating Environment (OE), solaris is used as a prefix and what is being authorized is indicated as a suffix.
res1 - Reserved for the future
res2 - Reserved for the future
short_desc - A terse name for the authorization that is suitable for display in user interfaces
long_desc - A long description that identifies the purpose of the authorization, the application in which it is used, and the type of user who might be interested in it. It can be displayed as help text of an application.
attr - An optional list of semicolon-separated key-value pairs that describe the attributes of an authorization.

The *prof_attr* database fields are also separated by colons and described as below:
profname; res1; res2; desc; attr
profname - The name of the rights profile which is also used by the *user_attr* database to indicate assigned rights profiles.
res1 - Reserved for the future.
res2 - Reserved for the future.
desc - A long description that explains the purpose of the rights profile.
attr - An optional list of semicolon-separated key-value pairs (help and auths) that describe the attributes to apply to the object on execution.

The fourth database is the *exec_attr* database with fields as below:
name; policy; type; res1; res2; id; attr
name - The name of the rights profile in the *prof_attr* database.
policy - The security policy that is associated with this entry. (Valid policy is 'suser').
type - The type entity that is specified (valid type is *cmd* (command)).
res1 - Reserved for the future.
res2 - Reserved for the future.
id - A string that identifies the entity.
attr - An optional list of semicolon-separated key-value pairs (euid, uid, egid, and gid) that describes the security attributes to apply to the entity on execution. The list depends on the policy enforced.

In addition to these four databases, the *policy.conf* database is also important for the implementation of RBAC. Figure 4.1 shows the relationship of these databases fitted in our earlier generalization of RBAC authorization database in Chapter 3.

4.3.3 Planning for Solaris RBAC

The Sun Microsystems documentation [31] recommends a thorough knowledge of the RBAC capabilities as well as the requirements of the organization. The

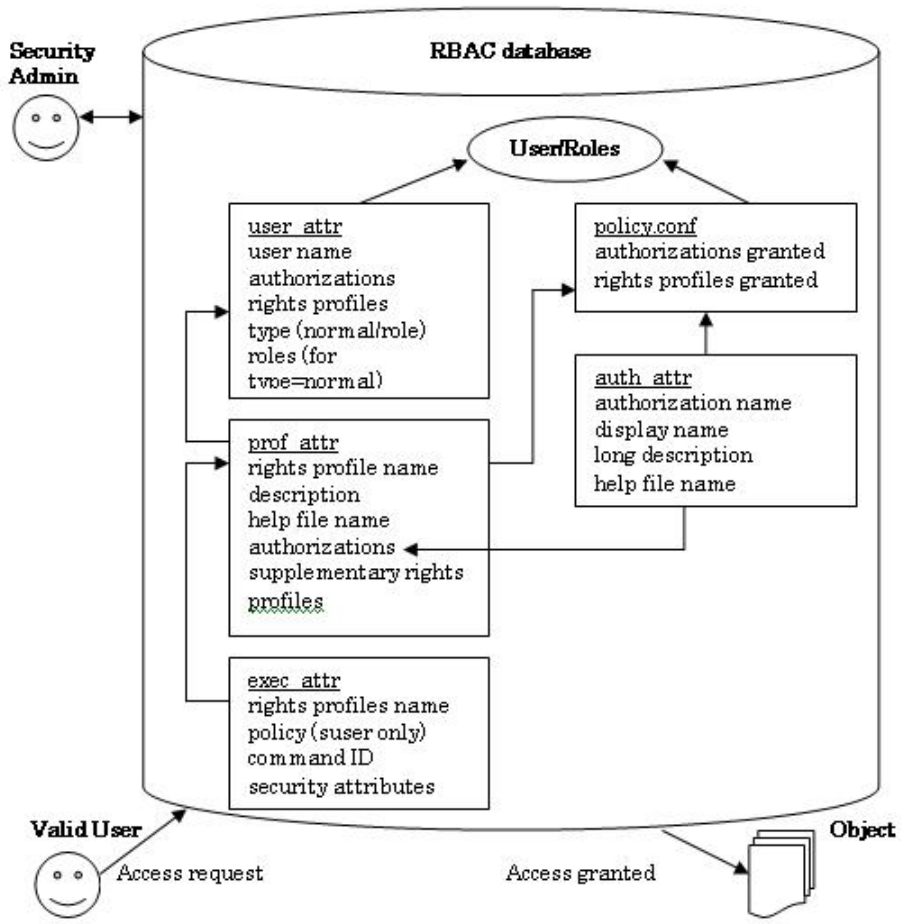

Figure 4.1: Solaris RBAC database

following steps together with the master plan in 4.3.1 are recommended when planning for a Solaris 9 RBAC implementation.

1. It is essential to learn the basics of RBAC before its implementation. Solaris documentation gives enough background and literature on RBAC.

2. There should be a security policy in place that details potential threats to the system, measure of the risk of each threat, and a strategy to counteract these threats. It is a good idea to examine the security policy in place and customize RBAC configurations to hold on it.

3. Depending on the organization's security needs, decide how much RBAC is required. The following are the forms of RBAC that can be configured:

   - *No RBAC* - all tasks performed as root user, where users log in as themselves and when they select a console tool they type root as a user.

- *Root as Role* - to eliminate anonymous root logins all root users must login as themselves and assume the root role.
- *Single Role only* - Primary administrator role is the only role added as an alternative to superuser model.
- *Suggested Roles* - For organizations with administrators at different levels of responsibility whose job capabilities fit suggested roles (Primary Administrator, System Administrator, and Operator).
- *Custom Roles* - depending on the security requirements of the organization, specific roles are configured.

4. Decide which suggested roles are appropriate for the organization.
5. Decide on additional roles or rights profiles which are appropriate for the organization.
6. Identify users that should be assigned available roles.

In response to the recommended steps, the basics of RBAC presented in [[8], [30], [31], [32]] on a Solaris RBAC Implementation, were studied, and the MIS department's immediate degree of RBAC required at that moment was identified as making 'root a role' to eliminate anonymous root logins on the Solaris 9 servers. Though all the three suggested roles (Primary Administrator, System Administrator and Operator) are important for the MIS department, it requires time for users to understand RBAC. We made a survey in MIS department and determined users who required root privileges on the appropriate servers. These are the users that originally had root privileges but could login anonymously.

4.3.4 Server Installation and Role configuration

In order not to tamper with existing servers, a new server was installed specifically for the project. This was to help the team to understand how RBAC works before it is migrated to the existing systems. After server installation, root was made a role. The standard operating procedure (SOP) was prepared tested and formalized, and the configuration was migrated to one of the nine Solaris servers in place. The details of these documents were made in a separate confidential report to the Institution.

4.4 Before and After Root Role Configuration

In this section we discuss the RBAC database before and after the configuration of root as a role, and we give general consequences of configuring RBAC features on the system.

4.4.1 RBAC and Authorization Management

While the Solaris 9 OE was already in use, its RBAC features needed to be fully exploited. There were no roles configured, instead system accounts were also regarded as normal users. Before making root a role, there were about five root users all logging on to the system directly as root. As a security measure, root password storage was entrusted with one individual who would change it

regularly and give it to a user on signing for it as to when it is required. All other system accounts were handled in the same way.
An excerpt from the *user_attr* database looked as below:

```
root; ; ; ; type=normal; auths=solaris.*, solaris.grant ; profiles=All
lp; ; ; ; type=normal; profiles=printer management
adm; ; ; ; type=normal; profiles=log management
user1; ; ; ; type=normal
........
user(N-1); ; ; ; type=normal
userN; ; ; ; type=normal
```

N being the number of normal users in the database.
From the excerpt above, Solaris root user had all solaris authorizations and all rights profiles, and could grant any to other users. This means any root user had all powers to read and write to any file, run all programs, and send kill signals to any process. Any root user, could do any thing to any confidential file including financial records in the Institution, could even shut down the entire network. Moreover root users would be able to accomplish their actions without their real UIDs being available for auditing.

The essence of RBAC as seen in Chapter 3 is to make sure that no user is given more privilege than necessary for performing his or her job. By leaving root user with all these powers, the principles of least privilege and separation of duty are contravened. By making root a role and creating other special accounts (roles), which could be assigned some of root's capabilities reduces the risk of compromising security.

Using root as a normal user was not only a security risk but also an inconvenience to users. A simple example is when the person entrusted with root password would be absent by any reason, and not available on phone, no task would be performed by any root user. This would affect other services including performance. After making root a role, root users login using their UIDs and then assume the role *root*. If any user accidentally or maliciously performs undesirable action, the action is traced back to the individual who assumed the role to perform the task. This traceability allows root password to be easily shared among the identified root users to perform their tasks when need arises.
An excerpt from the *user_attr* database after making root a role looked as below:

```
root; ; ; ; type=role; auths=solaris.*, solaris.grant ; profiles=All
lp; ; ; ; type=normal; profiles=printer management
adm; ; ; ; type=normal; profiles=log management
user1; ; ; ; type=normal; roles=root
user2; ; ; ; type=normal; roles=root
user3; ; ; ; type=normal; roles=root
user4; ; ; ; type=normal; roles=root
user5; ; ; ; type=normal; roles=root
user6; ; ; ; type=normal
........
user(N-1); ; ; ; type=normal
userN; ; ; ; type=normal
```

From the excerpt above, root has authorizations *solaris.** and *solaris.grant*. It means that root role still has all powers and is permitted to delegate to other users any solaris authorization. This is still dangerous, since any of the five users assigned to root has such powers and can do anything with the system.

The 'All' profiles include even those assigned to other accounts such as printer management and log management profiles. These accounts can be made roles and their capabilities removed from those of root. But the MIS department is not yet ready for such abrupt changes, and requires time to get prepared.

Nevertheless, with current configuration, root users can perform their tasks without any inconveniences and can be accountable for their actions. It is also good to note that existing root users can always be changed by modifying their RBAC properties according to their new assigned tasks and new users can also be added as need arises without tempering with authorization rules of root. This makes management of authorizations easy.

However, the flexibility in the Solaris 9 OS that gives customers freedom to configure or ignore security features such as RBAC need to be revisited. If alert mechanism were embedded in the system, may be they would make the product customers aware of security features included. This would mean that the system would not allow being operational unless all its basic security features are configured. The RBAC feature is an important one that need not be ignored in system security, but the system can still serve with or without RBAC's configuration. Such an overlook on the part of the developer needs to be addressed in case the system is to be evaluated based on security benefits such customers have gained from it.

4.4.2 Consequences of RBAC on Security properties

The consequences of configuring root as a role on security properties are summarized in Table 4.1

4.5 Recommendations

1. An access control policy to enhance the general security policy should be written.

2. It is recommended that the Solaris 9 OE RBAC features are studied, understood and full implemented in order to utilize RBAC for good security practices.

3. Configuring suggested roles (System Administrator and Operator) in Solaris 9 RBAC to relieve root role of some of the capabilities will meet the principle of least privilege, and also give the MIS department a chance to appreciate RBAC values and benefits.

4. It is also recommended that in future the Institution comes up with an RBAC system depending on its security needs. One example would be a simple system with three roles for staff to access information. The roles can be:

    - Permanent
    - Contract
    - Temporary

The Institution has permanent, contractual and temporary staff, it is not advisable to allow all of them to access documents on the LAN with equal privileges. Different roles can be given different read, write privileges on documents according to their tasks.

5. In relation with the preceding recommendation, such RBAC system would take into consideration other operating systems such as Windows.

6. It is recommended that other special roles especially on applications running on Solaris 9 systems such as 'oracle user' on Oracle Database Management Systems, be configured to reduce security administrative costs.

7. Upgrading Solaris 9 OS to Solaris 10 (Solaris 10 already exists but still in testing phase) is recommended for all servers given the additional security features in the later. However, administrators require more training and practice with command line tool since the 'admintool' which provides graphical user interface is no longer recommended for security purposes.

8. It is recommended that the Research department together with the MIS department periodically formulates research projects in any desirable information technology field relevant to information systems of the Institution, to take advantage of research students seeking internship. This would help the Institution keep up-to-date with technology changes at the same time doing routine work. MIS staff would also benefit a lot in supervising students on projects formulated by them.

9. It is recommended that the Institution's Library together with MIS department identifies and subscribes to computer and information security journals that could be of great use for MIS staff interested in research and new technologies for their day to day activities.

10. Another recommendation is that the Institution should always seek maximum utility of any product purchased for the good of business continuity.

The benefit of having operating systems and applications' accounts well managed is that they are a backbone of good security practices in any computerized system. Many organizations have up-to-date software, hardware, and technologies, but lack the expertise and knowledge to utilize them for intended purposes.

4.6 Conclusions

This chapter has highlighted major factors influencing the access controls used by the Institution in the case study and presented a plan for implementing RBAC in the Institution. The chapter also discussed the steps for implementing the Solaris 9 OS RBAC features as a means of security control to restrict insiders' when using system resources. The effect of making root a role, on the maintainability of RBAC related databases is presented with excerpts of the contents of the database before and after the configuration. The chapter has also summarized the benefits of the configuration in relation to other security properties. Recommendations thought to be useful for the Institution have been

given. An observation, that there is a need to develop systems with extra features to alert users about the security mechanisms embedded in systems has been made for the developers' attention.

We believe this chapter will act as an eye opener to both the developed and developing world that there is a lot to understand about access control technologies which we think is not exhausted in this case study. We are convinced that this is just the beginning; more case studies are required in achieving secure systems. As much as it is good to research for newer technologies to replace the old ones, it is also important to understand why the old ones have not been easily adopted and used. This will give balanced research as we strive for adequate information security.

Table 4.1: Consequences of Root Configuration

Property	**Before Root Configuration**	**After Root Configuration**
Confidentiality	Preventing unauthorized disclosure of information could not be totally achieved with anonymous root logins.	Root users are aware of their UIDs being audited, cannot easily disclose information anyhow. The principle of least privilege is not yet fully met.
Integrity	Like in confidentiality, users could hide under anonymous logins and modify information.	Same as above.
Authentication	As long as a user has obtained root pass word whether rightfully or by any other means, would be regarded as a root user and allowed access.	Unless a user is assigned root role and authenticated as so, cannot access resources authorized for root role. More so, no direct logins are allowed even if some one root password.
Authorization	Any user with root password had all authorizations for root capabilities. In other words, it is a user account assigned profiles but not a role account.	Unless a user assumes a role account to gain authorizations to perform role tasks, he or she can access root capabilities.
Access Control	General access control to root account done by changing the password regularly and making users sign for the password.	Added on another control by monitoring individual root user and auditing his or her UID to make sure that he has done what he is authorized to do.
Accountability	Accountability stopped at auditing root account but not an individual user.	The user who assumes root role is accountable for his actions during the session.
Performance	Manual authorization hinders performance especially the password storage business. Suppose the person with password is absent. Work stops.	Automated administration of authorizations can maximize operational performance as well as system performance. Root users are provided with what they require for their work, though the privileges are more than enough at the moment.
Cost	Cost is relatively high in terms of an OS which is in place and already paid for but not fully utilized.	Some cost benefits are being realized in terms of performance. But to be appreciated, all RBAC features must be configured and utilized.
Availability	Human errors occur with or without RBAC, but manual handling of root password may cause errors that easily affect availability of the system.	RBAC minimizes errors. Only those automatically authorized for root role can perform root tasks with appropriate rights.
Manageability	Complex	Still hard, until RBAC is fully implemented.

Chapter 5

Conclusions and Future Work

This chapter summarizes the contributions of the research and reflects what has been established on role based access control (RBAC) using literature research approach coupled with a case study in a financial institution in Uganda. This thesis appreciates the level of research that has already been done on access control in general and RBAC in particular. However, it observes that focusing on new approaches alone cannot improve security of systems especially in Uganda. Reflecting on old approaches and their benefits and liabilities, and redefining some concepts, as new ones are studied can improve the situation.

The research observes a number of benefits in using RBAC:

- RBAC eases security administration since a user gets privileges through a role. Administrators are much more concerned with privileges required for a role to a certain extent than those needed by an individual user to perform tasks. With modern organizations where flat as opposed to hierarchical structures are encouraged, there are typically fewer roles than individual users. Moreover, in this era of e-services, it is obvious that roles are likely to be much fewer than participants.

- RBAC simplifies privileges management: Privileges are assigned to roles and roles really change, but individuals change roles frequently so that roles are occupied by different people at different times. Administrators' main task is to assign and de-assign roles to individuals.

- RBAC supports separation of duty. An individual user that initiates a process or a transaction is constrained not to carry it on up to completion if there is conflict of interest. Like wise a user cannot be assigned two conflicting roles. This makes RBAC instrumental in commercial sector to prevent fraud.

- Role hierarchy provides a flexible way to inherit privileges. Superior roles in the hierarchy inherit least privileges assigned to junior roles.

- The principle of least privilege prevents Trojan horse problems. Users are

restricted to roles that they need for their jobs, and roles are also restricted to operations that are required for their tasks nothing more.

Some aspects that require improvements are also highlighted:

- There is a possibility of an attack on the mappings of a user to a role, which require more investigation, especially if the minimum privileges available in a user's session are enough for the attack to be accomplished.
- Though the separation of duty principle is supported by restricting an individual user from taking up two conflicting roles, the user can assume a third role in a related area. It may be possible for the third role assumed, to give the user clues of actions between the former roles hence contravening the principle.
- Flexibility is still required for a user to be able to assume a role with some of the role's privileges but not all at once. For example, in Solaris RBAC there is no way for a user to assume some root privileges but not all at once.

Summary

In Chapter 2 we defined access control as a process of limiting right of use of objects in an information system to only authorized subjects by enforcing specified rules. We also discussed factors that any organization must consider when setting up an access control system as access control policies, mechanisms and models. We observed that, the traditional mechanisms (MAC and DAC) have some deficiencies such as complexity in maintaining the authorization database, rigid rules which are undesirable in commercial organizations and Trojan horse problems.

In Chapter 3 we discussed RBAC as an alternative model for access control to address the deficiencies of MAC and DAC using the concept of roles to grant privileges on objects. RBAC is policy neutral and can be implemented in any sector. We redefined privileges and authorizations in a simple way for new readers interested in understanding RBAC. Our contribution in chapter 3 is putting together basic information related to RBAC for those who are new to the approach especially in Uganda. The chapter also acted as a base to evaluate access control implementation in the case study.

In chapter 4 we presented our case study in Uganda. We were interested in evaluating security of a system based in a 'less-developed' environment using role based access control. As a contribution, our study highlighted the following, for consideration:

- An access control policy may not exist in some organizations despite crucial information they may handle.
- Lack of knowledge and expertise, limited availability of literature on ongoing research and limited financial resources among others, still hinder 'developing countries' from having secure access control systems in place.
- Most literature is based on research conducted in developed world, but our approach revealed that a lot need to be done to include developing world in this Information era. The physical environment that systems

work in, especially the people and their organizational structure and culture is taken into consideration. We argue that it is essential to discuss information and system security with a good understanding of culture in a society involved in a particular system.

- However much access control mechanisms seem to be practically embedded in most systems; they are not properly implemented by customers and users.
- System developers may need to think of incorporating mechanisms in systems to alert users about configuring security features during installation period. For example Solaris 9 OS would be able to alert the user about RBAC features on installation, so that at least the minimum of its features is utilized.

Another contribution of the case study is that an organizational plan to implement RBAC has been set and as an immediate result the Solaris 9 RBAC features are configured on Solaris nine servers which has eased administration of root's privileges.

Future Work

Future work may include but is not limited to the following:

- Compare access control systems within the framework or formalization.
- Design and implement an RBAC related model to fit in the situation of countries like Uganda.
- Address the concerns highlighted in RBAC.
- Is RBAC good for organizations which do not have defined organizational structures in place?
- Though RBAC appear to have most of the solutions to access control problems there is need to find out why it is not practically replacing traditional access control methods as fast as possible.
- The principle of roles is already in use as far as e-service systems are concerned, and seem to have made privilege management easy, but the information structure and communication language between the client roles and service provider roles need to be made secure.
- Deeper understanding why the least developed and the developing world is hesitant to adopt and implement security mechanisms embedded in systems while they are adopting digital solutions. Can they participate fully in Information Technology revolution?

We are convinced that this is just the beginning; more research and case studies are required in achieving secure systems globally. Because of its flexibility, and ability to reduce security flaws, RBAC may easily be understood and adopted by many organizations in Uganda, if well explained and presented in a simple way. As much as it is good to research for newer technologies, it is also important to understand why the old ones have not been easily adopted and used. This will give balanced research as we strive for adequate information security.

Bibliography

[1] R. Adaikkalavan and S. Chakravarthy. A framework for supporting and enforcing rbac and its extensions in a seamless manner, 2001. NIST Special Publication 800-33.

[2] M.A. Al-Kahtani and R.S. Sandhu. Induced role hierarchies with attribute-based rbac. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 142–148, New York, NY, USA, 2003. ACM Press.

[3] APACHE. Apache http server version 1.3. authentication, authorization, and access control. `http://httpd.apache.org/docs/1.3/howto/auth.html`.

[4] R. W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *IEEE Symposium on Security and Privacy*, pages 116–132, 1990.

[5] S. Berinato. The global state of information security, 2005. CSO Magazine, CXO Media Inc.

[6] E. Bertino, E. Ferrari, and E. Pitoura. An access control mechanism for large scale data dissemination systems. *ride*, 00:0043, 2001.

[7] D.F.C. Brewer and M.J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[8] M.T. Chalfant. Role based access control and secure shell: A close look at two solaris operating environment security features, 2003. Sun BluePrints Online.

[9] S. Chou. Lnrbac: A multiple-levelled role-based access control model for protecting privacy in object-oriented systems. *Journal of Object Technology*, 3(3):91–120, 2004.

[10] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. *sp*, 00:184, 1987.

[11] D.D. Downs, J.R. Rub, K.C. Kung, and C.S. Jordan. Issues in discretionary access control. *sp*, 00:208, 1985.

[12] E.B. Fernandaz and G. Pernul. Patterns for session-based access control, 2006. `http://www-ifs.uni-regensburg.de/fileadmin/Forschung/PDF_Publikationen/AccCtPattSept2206.pdf`.

[13] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th (NIST)-(NCSC) National Computer Security Conference*, pages 554–563, 1992.

[14] D.F. Ferraiolo, J.F. Barkley, and D.R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur.*, 2(1):34–64, 1999.

[15] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[16] R. Ferrairo, J. Cugini, and R. Kuhn. Role based access control (rbac): Features and motivations. In *Proceedings, Annual Computer Security Applications Conference*. IEEE Computer Society Press, 1995.

[17] R. Focardi and R. Gorrieri, editors. *Foundations of Security Analysis and Design, Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*. Springer, 2001.

[18] Center for Technology in Government. Internet security seminar. Technical report, Universuty at Albany, 1996.

[19] Alliance for Telecomunications Industry Solutions (ATIS). Atis telecom glossary 2000. `http://www.atis.org/tg2k/`.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, Inc., Boston, MA, USA, 2002.

[21] A. Haddad. Meca: a tool for access control models. In Jacques Julliand and Olga Kouchnarenko, editors, *The 7th International B Conference (B'2007)*, volume 4355 of *LNCS*. Springer-Verlag, 2007.

[22] V.C. Hu, D.F. Ferraiolo, and D.R. Kuhn. Assessment of access control systems. Technical report, National Institute of Standards and Technology, 2006.

[23] T. Jaeger, R. Sailer, and Y. Sreenivasan. Managing the risk of covert information flows in virtual machine systems. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 81–90, New York, NY, USA, 2007. ACM Press.

[24] W. A. Jansen. Inheritance properties of role hierarchies. In *Proc. 21st (NIST)-(NCSC) National Information Systems Security Conference*, pages 476–485, 1998.

[25] C.S. Jordan. A guide to understanding discretionary access control in trusted systems. Technical Report Library No.S-228, 576, National Computer Security Center (NCSC), Fort George G. Meade, Maryland, 1987.

[26] D. Kienze, M. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns template and tutorial, 2002.

[27] S.R. Kodituwakku, P. Bertok, and L. Zhao. Aplrbac: A pattern language for designing and implementing role based access control, 2001. In Proceedings of the Euro-PLoP2001.

[28] D.R. Kuhn. Role based access control on (mls) systems without kernel changes. In *ACM Workshop on Role-Based Access Control*, pages 25–32, 1998.

[29] PC Magazine. Definition of: mandatory access control. `http://www.pcmag.com/encyclopedia_term/0,2542,t=mandatory+access+control\&i=46573,00.asp`.

[30] Sun Microsystems. Rbac in solaris operating environment: Sun microsystems white paper, 2001.

[31] Sun Microsystems. Solaris 9 9/04 system administrator collection, system administration guide: Security services, 2003. Sun Microsystems, Inc. Santa Clara, CA, USA.

[32] A. Noordergraaf and K. Waston. Soralis operating environment security. updated for soralis operating environment, 2002. Sun BluePrints Online.

[33] M. Nyanchama and S.L. Osborn. Access rights administration in role-based security systems. In *(IFIP) Workshop on Database Security*, pages 37–56, 1994.

[34] T. Priebe, E.B. Fernandez, J.I. Mehlau, and G. Pernul. A pattern system for access control.

[35] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control: A multi-dimensional view. In *10th Annual Computer Security Applications Conference*, pages 54–62, 1994.

[36] R.S. Sandhu. Role hierarchies and constraints for lattice-based access controls. In *ESORICS '96: Proceedings of the 4th European Symposium on Research in Computer Security*, pages 65–79, London, UK, 1996. Springer-Verlag.

[37] R.S. Sandhu, V. Bhamidipati, and Q. Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.

[38] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C. E. Youman. Role-based access control models. *(IEEE) Computer*, 29(2):38–47, 1996.

[39] R.s. Sandhu and Q. Munawer. How to do discretionary access control using roles. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 47–54, New York, NY, USA, 1998. ACM Press.

[40] R.S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[41] A. Schaad, J. Moffet, and J. Jacob. The role-based access control system of a european bank: A case study and discussion, 2001. In Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001.

[42] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns : Integrating Security and Systems Engineering (Wiley Software Patterns Series)*. John Wiley & Sons, March 2006.

[43] A. Spalka and H. Langweg. Notes on application-orientated access control. *dexa*, 00, 2002.

[44] C. Steel, R. Nagappan, and R. Lai. *Core Security Patterns. Best Practices and Strategies for J2EE, Web Services, and Identity Management.* Pearson Education, Inc., 2005.

[45] G. Stoneburner. Underlying technical models for information technology security, 2001. National Institute for Standards and Technology SP 800-33.

[46] Wikipedia. Access control. `http://en.wikipedia.org/wiki/Access_control`.

[47] Wikipedia. Access control. `http://en.wikipedia.org/wiki/Biba_Model`.

[48] J. Yoder and J. Barcalow. Architectural patterns for enabling application security, 1997. In Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97), 1997. 2.