Formal Security Analysis of Access Control Models and Their Spatiotemporal Extensions

by
Emre Uzun

A dissertation submitted to the
Graduate school-Newark
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Management
Information Technology Major

Written under the direction of
Dr. Vijayalakshmi Atluri
Dr. Jaideep Vaidya
and approved by

_____

_____

_____

_____

_____

Newark, New Jersey
October 2015

DISSERTATION ABSTRACT

Formal Security Analysis of Access Control Models and Their Spatiotemporal Extensions

By Emre Uzun

Dissertation Directors: Dr. Vijayalakshmi Atluri and Dr. Jaideep Vaidya

Providing restrictive and secure access to resources is a challenging and socially important problem. Today, there exists a variety of formal security models to meet the wide needs of requirements in specifying access control policies. These include Discretionary Access Control (DAC) and Role Based Access Control (RBAC). For every model, it is necessary to analyze and prove that the system is secure, or in other words, access rights of sensitive data are not leaked to potentially untrusted users (rights leakage), as well as the data itself (data leakage). Analysis is essential to understand the implications of security policies and helps organizations gain confidence on the control they have on resources while providing access, and devise and maintain policies. There is a dire need for such analysis tools that help security administrators as they make administrative changes to reflect changes in policy.

In this dissertation we tackle two major problems: Rights leakage problem and data leakage problem. For the rights leakage problem, we focus on RBAC

and its temporal and spatiotemporal extensions, since RBAC has been successfully incorporated in a variety of commercial systems, and has become the norm in many of today's organizations for enforcing security. Towards this end, we first propose suitable administrative models that govern changes to policies. Then we develop efficient security analysis techniques and tools, in which we explore a decomposition strategy, that splits the temporal or spatio temporal security analysis problems into smaller and more manageable sub-problems which in fact, are RBAC security analysis problems on which the existing RBAC security analysis tools can be employed. We then evaluate them from a theoretical perspective by analyzing their complexity, as well as from a practical perspective by evaluating their performance using real world and simulated data sets.

For the data leakage problem, we consider two types of data leakages: confidentiality violating and integrity violating. In confidentiality violating data leakage, sensitive data in an object can be leaked to potentially untrusted users via another object that is readable by those users. In integrity violating data leakage, on the other hand, data can be leaked to an object where the user is not allowed to write to explicitly. We propose techniques to eliminate these possible leakages by using three different strategies: Conservative, Proactive and Retrospective. We then computationally evaluate them to show the running times and restrictiveness of our proposed methodologies in terms of identifying the possible data leakages and eliminating them.

PREFACE

Dissertation Committee Members

- Dr. Nabil Adam, Rutgers University
- Dr. Vijayalakshmi Atluri, Rutgers University
- Dr. Madhusudan Parthasarathy, University of Illinois at Urbana-Champaign
- Dr. Shamik Sural, Indian Institute of Technology Kharagpur
- Dr. Jaideep Vaidya, Rutgers University

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my family for their everlasting love and support. My studies in the United States would not have been realized without them. I would like to express my sincere gratitude to my supervisors Dr. Vijay Atluri and Dr. Jaideep Vaidya for their invaluable guidance, encouragement and motivation during my doctoral study. They have been always ready to provide help, support and trust with everlasting patience and interest. I have learned a lot of things from them, not only in academic but also in personal and intellectual matters.

I would like to thank to my committee members Dr. Nabil Adam, Dr. Shamik Sural and Dr. Madhusudan Parthasarathy for accepting to read and review this dissertation and their substantial comments and suggestions. Also, thanks to Dr. Gennaro Parlato and Dr. Anna Lisa Ferrara for their valuable help and support.

It is a pleasure for me to express my deepest gratitude to my friend Dr. Sıtkı Gülten for being so patient, helpful and considerate during my entire graduate study the second time. Thanks to my friends in New Jersey and across Hudson River for their priceless friendship for the last 5 years.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

Security in computer systems gains significant importance as more processes in enterprises become digitized. One of the basic ways to provide security is to impose access control over the resources especially if they are shared among various different users in a large enterprise. In the literature there are mainly three different models – called access control models – to achieve this target: Discretionary Access Control (DAC), in which the permissions on the objects are assigned to the users directly and the users have full control over the objects that they own [44]. Mandatory Access Control (MAC), in which objects and subjects are assigned security labels indicating their clearance levels. Each user can read the objects at their security level and lower levels, and can write the objects at their security level and higher levels. Although DAC and MAC provide security in access control, it is likely for these models to have huge operational costs. For instance in DAC, when a new user is introduced into the system, all permissions that user requires must be granted one by one. This operation can be handled in small systems, but in large systems where the number of permissions are measured in thousands, even a minor permission update operation is likely to take a huge amount of time. In MAC, on the other hand, the access control levels are determined solely by the security labels of the users and objects. Consider a new user (or object) that

should be in a security level in between two previously defined consecutive security levels. Then, for each of these situations a new security level must be created and the security level of other users and possibly other objects should be updated. Hence, in order to overcome these type of drawbacks without sacrificing any of the security properties of DAC and MAC; Role Based Access Control (RBAC) model is developed. The core of this access control model is based on the formation of roles, or can be called permission groups, which usually represent job functions in an enterprise. Users in the system are assigned to roles instead of being directly assigned to the permissions. The underlying reasoning is that, users are generally assigned to a set of permissions that represent their job function and the permissions required for a job function rarely change with respect to the users that perform that particular job [44]. This property immediately reduces the operational costs of the system, since the number of roles is usually smaller than that of the permissions. Furthermore, despite being similar to RBAC, there are some other models proposed like Task Based, Team Based, and Coalition Based Access Control to address different organizational requirements. [15, 23, 37, 48, 49, 55]

The benefits of RBAC lead to development of some useful extensions. In particular, researchers preserve the basic idea of having roles in the model and add some additional layers, like time and space. Temporal RBAC (TRBAC) [11], Generalized Temporal RBAC [33], Spatio-Temporal RBAC [2] are some examples to these extensions.

1.1 Leakage of Rights and Data – Importance of Security Analysis

As providing access control, which is one of the core requirements of security in computer systems, the security properties of the system must be verified in order to ensure that there would not be any violation of the security policies. By violation, we mean any activity that causes unauthorized users to gain access to sensitive data. These violations, which we denote as *leakages* can be of two types: The *access control rights* of a confidential object can be leaked, so that the unauthorized subjects gain explicit access to these objects (rights leakage), or the confidential *data* itself can be leaked to other objects so that the unauthorized subjects implicitly gain access to these objects (data leakage). In the former case, the *rules* of the *administrative model*, which governs modifications on the access control rights, are abused to leak the rights of the objects containing sensitive information, whereas in the latter case, the configuration of access control rights allows users to leak data from a sensitive object to another object that can be accessed by users that are not authorized to access the sensitive object. In order to protect the confidentiality of the data and hence the security of the system, the access control policies, the administrative model and the data flow within the system must be checked thoroughly to decrease the likelihood of any type of leakage. This process is called *security analysis* and without which it is highly unlikely to determine whether the data is protected, because in any access control system, although each security policy look safe in isolation, their cumulative effect might create unforeseen violations.

1.2 Problem Statement and Contributions

There are numerous studies in the literature that seek to provide a good *security analysis*. Starting from the earliest study by Harrison, Ruzzo and Ullmann [26], the security analysis for rights leakage remains undecidable in the most generic case. Since then, researchers have been seeking to find new methodologies that would perform the analysis in a more reasonable time frame, while sacrificing some properties of the generic HRU model. Many of these succeeding studies propose techniques to decrease this complexity up to the point that it becomes decidable and tractable. Although the results obtained give valuable feedback about the level of security of an access control configuration up to some extent, the assumptions and reductions needed to achieve these improvements could yield an analysis that might not completely represent the real conditions as good as the generic HRU model, and this is obviously not desirable to attain full security. Hence, the ongoing dilemma between expressive power and computational complexity leads the discussion to the point that there is now a necessity for useful techniques that are built to perform security analysis with no or as few assumptions as possible no matter if the analysis is decidable or undecidable.

In this dissertation, we develop security analysis techniques to address the rights leakage and data leakage problems. For the *Rights Leakage Problem*, which seeks to determine whether a potentially untrusted user will ever get access to confidential objects, we propose security analysis methodologies that work for Temporal and Spatio-temporal extensions of the RBAC model. For the *Data Leakage Problem*, more commonly known as the Trojan Horse Problem, which seeks to identify and prevent any data leakage given the configuration of the access con-

trol system, we develop security analysis methodologies that work for DAC and RBAC models. In particular, our goals in this research are as follows: We identify and formalize the key policies and vulnerabilities in RBAC, including rights and data leakage, in the presence of temporal and spatio-temporal constraints. We aim to model the security problems for policies in access control as reachability problems in transition systems. We provide exact and heuristic strategies Finally we implement tools that realize the above techniques, and provide administrators with a usable and useful tool to express policies and automatically find breaches of their security policies. In summary, we have the following contributions in this dissertation. For the rights leakage problem,

- We propose an administrative model for Temporal RBAC and Spatio-temporal RBAC.
- In order to reduce the complexity of the model, we represent time as discrete and periodic intervals and location groups of physical locations – called logical locations.
- We develop a flexible security analysis methodology based primarily on decomposing the TRBAC and STRBAC security analysis problems into smaller and more manageable RBAC security analysis problems.

For the data leakage problem,

- We investigate both confidentiality and integrity violating data leakages in DAC and RBAC models focusing on three different data leakage elimination strategies, namely, conservative, proactive and retrospective.

- We prove that the transitive closure is not necessary for the data leakage analysis. Instead, examining data flows among pairs of objects is sufficient for analysis purposes. This reduces the complexity of the problem significantly.

- We show that given an access control configuration obtaining a data leakage free access control matrix with minimal modification is an NP-Complete problem.

- We propose an Integer Linear Programming model for the conservative approach for DAC. We claim that the corresponding model for RBAC is likely to be non-linear, however it is still decidable.

- We propose more scalable heuristic alternatives for conservative approach.

1.3 Outline

The rest of the dissertation is organized as follows: In Section 2, we briefly summarize the related work done in the literature. In Section 3, we provide background information necessary to follow the models and analysis strategies proposed in the dissertation. In Section 4 and 5, we provide our research framework for rights leakage problem and data leakage problem, respectively. In these sections, more discussion and motivation is provided for the individual problems. Then, the proposed models and solution methodologies are provided along with experimental results. Finally in Section 6, we summarize our contributions on this dissertation and provide our future work.

CHAPTER 2
RELATED WORK

The pioneering works for the security analysis are done for protection schemes with Discretionary Access Control, which is usually composed of an access control matrix and some operators to modify the scheme. The earliest study is done by Harrison, Ruzzo and Ullmann (HRU) in 1976, which is the source of inspiration to the succeeding studies. The analysis of the HRU model proposed in this study leads to the first and the hardest obstacle to be overcome by the researchers in this area, which is the undecidability of the safety problem. The successors of HRU model, focus mainly on how to obtain a decidable safety analysis with minimum compromise to the original model, and yet none of these studies can provide a model that covers the full benefits and expressive power of HRU.

2.1 Security Analysis of Rights Leakage Problem on Discretionary Access Control Model

Harrison et al. [26] propose a formal model for protection systems, which is one of the earliest models in access control systems. They introduce the concepts of system configuration, safety problem and the undecidability of the safety problem. They show that there is an algorithm which decides whether or not a given mono-operational protection system and initial configuration is unsafe for a given generic right. However, it is undecidable whether a given configuration of a given

protection system is safe for a given generic right. It means that there does not exist any algorithm which can decide on the safety of an arbitrary protection system. On the other hand the safety problem is decidable (in polynomial space) for protection systems without create commands.

In 1978, Harrison and Ruzzo provide additional results on the conditions that affect the decidability of the HRU model (as depicted in [3]). As a result of this study, it is shown that safety is undecidable for bi-conditional HRU schemes (the *if* statements in the HRU commands have at least two conditions). However, it is decidable for mono-conditional schemes.

Jones et al. [29] provide a new protection system, called *Take-Grant Protection Model* which is based on graphs. This is actually a different representation of the protection model developed by Harrison et al. [26]. In the analysis, the key point is that, the access control rule set is not arbitrary, instead it is fixed which is generally true in computer systems. The main result is that the safety analysis of whether a user will gain an access right can be answered in linear time. On the other hand, keeping the rule set fixed is a drawback of the model. There can be many different types of rules, which should be included in the analysis in order to reflect the protection systems more clearly.

Sandhu [43] proposes the Schematic Protection Model in order to address the conflict between two objectives: generality of the access control model and tractability of the analysis in HRU and TGPM. The conflict is that, HRU is a very general model that has a high expressive power and covers many instances, but its analysis is intractable and even undecidable. On the other hand, the TGPM analysis is tractable but it considers very specific class of simple policies, so that

it may not be able to represent all possible scenarios. Hence, SPM is proposed to address these two objectives together to that they can be achievable together. The paper has two important assumptions related to the distribution of these components: (1) Acyclic Can Create: The entity creation operator *can create* can be seen as a directed graph, where nodes are entities and edges are the authorizations. Then, *can create* is said to be acyclic if there are no cycles in this graph. (2) Attenuating Create Rules: The child of a subject cannot have more rights than its parent. One final point is that revocations of rights are not allowed in the system, which makes it monotonic. Sandhu proves that SPM model has a high expressive power and provides a model whose analysis is both decidable and tractable, unlike the previously proposed HRU and TGPM, only if the model is acyclic and attenuating. Although SPM can be reduced to a TGPM and provides a better analysis than HRU and TGPM, SPM has some drawbacks related to being acyclic and attenuating, which makes it fail to represent the HRU model completely.

Ammann et al. [3] propose ESPM to address the limitations of SPM. In fact, the main outcome of the paper is that, it proves that ESPM is equivalent to HRU. The Extended SPM model is similar to SPM, with the difference of allowing multi parent creation option, which lacks in SPM. The remainder of the model is exactly the same as SPM. In the paper, the authors describe the multi parent modifications on the SPM and provide an algorithm for the security analysis of the acyclic and attenuating ESPM.

The benefits of having strong typing in the access control schemes as depicted in SPM model can also be embedded into the basic HRU model. Sandhu proposes Typed Access Matrix (TAM) model to address this issue and shows that HRU

is a special case of TAM. The paper applies the notion of strong typing to HRU schemes and yield the strong expressive power of HRU. It shows that strong typing is essential to obtain decidable safety analysis.

Soshi et al. [46] point out the deficiency of the TAM model of being decidable only when the system is monotonic. This assumption is not quite realistic, so that this model could lead to some problems when implementing to real life access control systems. Hence, the authors provide an extension of TAM, called Dynamic TAM (DTAM), in which changes in object types are allowed – therefore the strong typing assumption is broken. The authors show that allowing a non monotonic scheme and removing the restriction of strong typing can also provide a decidable safety analysis under certain conditions. However the algorithm provided is still intractable, meaning that there is no polynomial time algorithm to evaluate the safety of the nonmonotonic protection systems.

Overall, we can say that the generic safety question proposed by Harrison et al., is still undecidable and hence intractable. The monotonicity, strong typing and other assumptions sacrifice some of the expressive power of HRU, for the cost of obtaining a decidable safety analysis.

2.2 Security Analysis of Rights Leakage Problem on Role Based Access Control Model

Although it is structurally optimized to facilitate administration and regarded as a better alternative than Discretionary Access Control scheme, safety analysis for RBAC is as hard as DAC. In this section, we cover the most important safety analysis techniques proposed for RBAC.

Li and Tripunitara [35, 36] develop the first security analysis on RBAC. They provide the basics of the states, state transitions and queries that are required for the analysis. They develop two different classes of security problems for RBAC. The authors state that a general problem instance is decidable but intractable (coNP-Hard), whereas, if the query of interest is semi-static, then the problem can be solved in polynomial time.

Jha et al. [28] present a formal description of the security problem which is built on the URA97 module of the ARBAC97 administrative model. They define a Security Analysis Problem as whether there is a particular reachable state for the untrusted users possible under the current settings. The main result of the paper is that the security analysis problem on URA with a simple query of whether a user is a member of a particular role is PSPACE-Complete. Jha et al. [28] also demonstrate the performance of Logic Programming and Model Checking as tools to solve the Security Problem on URA. According to their results, the model checking approach scales better than the logic programming approach. The authors state that model checking can handle many different security queries in real life access control systems. The main difference between this paper and the papers by Li and Tripunitara [35, 36] is that in this paper there are simple queries and sophisticated state transitions, whereas Li and Tripunitara has sophisticated queries but simple state transitions.

Stoller et al. [47] considers analyzing the security problem in a parameterized complexity environment. More specifically, since the former studies about the security problem usually result in intractable (NP-Complete or PSPACE-Complete) complexities, fixing the parameter which causes this high complexity might re-

sult in a polynomial complexity in terms of the remaining variables. Stoller et al. perform studies on the ARBAC97 variant miniARBAC which only performs operations on URA, i.e. only has *can assign* and *can revoke* relations. They assume *Separate Administration*, which considers a system with users and administrators, so that none of the users can gain administrative privileges (i.e., the administrative roles and the regular roles are disjoint). This algorithm is said to be fixed parameter tractable with respect to the number of roles that appear as a positive and negative preconditions in some policies in the system. Moreover, they provide a slicing algorithm to further reduce the complexity.

Ferrara et al. [22] propose a set and numerical abstraction based reduction of ARBAC97 policies into programs, so that a program verification tool can be used to check the security properties. The most important distinguishing factors of their analysis is that they do not impose a separate administration assumption, so that they cover a more general setting; and they allow tracking multiple users at the same time. The former models can only handle one target user, however with the lack of separate administration, it is possible for regular users to gain administrative privileges, and hence they would have the rights to assign roles to other users. This brings out the necessity to track more than one regular user in the system to see the possible interactions among them. The analysis proposed in the paper is an abstraction of the RBAC system, so that this abstraction yields a sound analysis. In particular, when the analysis results that a specific role is not reachable in the abstraction, then it is not reachable in the real system. Whereas, if the analysis results that a role is reachable, than it cannot be concluded that whether that role is reachable or not in the real system. According to the results

they obtain, the model scales well to analyze security properties of large ARBAC policies.

2.3 Security Analysis of Rights Leakage Problem on Spatial and Temporal Extensions of RBAC Model

Even though RBAC is a powerful scheme to be implemented in various real life access control systems, it may fail to capture all of the details of a system. For instance if the access control policies of an enterprise change for different shifts during a day, then it is impractical to represent this system with an RBAC and changing the assignments manually in every different shift. Therefore, it is acceptable to have an additional temporal layer built into the model. Similarly, there could be some spatial constraints that enforces different access control policies in different locations. There are some research done for the extensions of RBAC in temporal and spatial dimensions.

The first model that embeds temporal data to access control is proposed by Bertino et al.[10] and called the Temporal Authorization Model (TAM). The model is basically built on the Discretionary Access Control model using discrete time. The model has temporal notions embedded into access control for both validity of authorizations and temporal dependencies among the authorizations. The model also have some basic control of separation of duty with the rules.

The study by Gal and Atluri [5] is another model that embeds the temporal notion into access control. The motivating factor for this is that every piece of data has a time which denotes the time that it is *captured* and the time interval in which it stays *valid*. It is a general fact that in real life, many applications require

this type of temporal notion, such as daily sales data or financial data. In order to capture this temporal notion, Gal and Atluri propose the Temporal Data Authorization Model (TDAM) that is capable of covering this type of temporal data. The TDAM model is a complementary model to TAM, since in TAM, administrators should specify the temporal notion in the system. However, TDAM derives this data directly from the attributes of the data.

The first temporal model developed on RBAC is proposed by Bertino et al. [11]. They propose Temporal RBAC with periodic role enabling and role triggers, which cause roles to be enabled and disabled. Furthermore, their model is capable of handling some *run time* enabling disabling requests. According to Joshi et al. [33], the work by Bertino et al. cannot handle some other temporal concepts like temporal user to role and temporal permission to role assignments. Further, the state of the roles (enabled - disabled) described by Bertino et al. is not sufficient to represent the real life situations. Also, one should distinguish between whether a role is enabled and a role is assumed. In their paper Joshi et al. [33] proposes Generalized Temporal RBAC model which considers the following in addition to the Temporal RBAC by Bertino et al.: Temporal constraints on role assignments, role activations, enabling and disabling constraints (like cardinality constraints), and temporal role hierarchies and SOD constraints.

Atluri and Chun [4] propose the first study that captures the spatial layer into the access control. More specifically, they propose an authorization model called Geo-Spatial Data Authorization Model (GSAM) which is capable of making authorization decisions with respect to the spatial and temporal attributes of the data. Aich et al. [2] propose STARBAC, which combines spatial and temporal layers

and embeds them into RBAC. Apart from the work by Atluri and Chun, the model described in this paper is an RBAC extension. In the paper, role enabling and disabling processes are controlled by space and time constraints. The space dimension is expressed as Logical Locations rather than Physical Locations (as in [4]), so a more general setting which allows different logical locations at the same physical location (such as different departments in the same building) is achievable. The paper also provides *Role Control Commands* which are logic based formal expressions to write the spatio-temporal access control policies.

Mondal et al. [40] provide a security analysis for Generalized Temporal RBAC using timed automata. In particular, they show the reduction of the temporal properties of a GTRBAC model, which include temporal role enabling and activation, role triggers and constraints, to a timed automata model. They perform an analysis using the timed automata models to verify the safety and liveness security properties of a GTRBAC system. This real time verification process is PSPACE-Complete. The important observation is that the verification process has a state space explosion for large number of users. In fact, the trade-off is whether to have large number of temporal constraints, or large number of users, roles and permissions.

2.4 Security Analysis of Data Leakage Problem

The data leakage problem is referred as the confinement problem or Trojan horse problem in the literature. The problem arises because of the drawbacks of the DAC (and RBAC) being incapable of governing the flow of the data, in addition to access to the data. The problem is defined in slightly different ways by dif-

ferent researchers. Lampson [34] defines confinement to enforce restrictions on programs to transmit data only to its caller by means of having total isolation. The problem, as discussed in [13, 24, 27], is the unauthorized flow of data between two objects in a computer system that would allow subjects to access sensitive data they cannot explicitly allowed to access. Gasser [24] also mentions about the alterations of access control rights as a result of Trojan horse attacks. Furthermore [38, 60] considers the Data Leakage Problem to gain implicit write access to objects that one does not have explicit write access.

Although there are numerous studies proposed to address this problem, considerably the most famous one is the Mandatory Access Control model (MAC). MAC is developed to prevent any data leakage from occurring. The most commonly known MAC is the Bell-LaPadula Model [9]. In MAC, every subject and object is attached a security label indicating how confidential the object and how authorized the subject is. For instance, these labels can be Low, Medium, High, for both subjects and objects. Then, the access control decisions are given in the following manner: Any subject has a read access to the objects that are on the same level or lower levels; a write access to the objects that are on the same or higher levels. Then, a subject cannot leak data from an object to other objects in the lower levels, and hence to the subjects in the lower levels.

Most of the remaining other studies in the literature make use of labels to determine access control. Boebert et al. [12], propose a domain and type based access control for military purposes. Badger et al. [6] propose a domain and type based enforcement to UNIX based computer systems to provide containing protecting and isolating information. Boebert and Kain [13] discuss the drawbacks of

the previous attempts to MAC and proposes another type based access control that basically follows the MAC perspective. Mao et al. [38] consider implementing a MAC-type control over a DAC system. In their model, every subject and object have a security label, in addition to the traditional DAC access matrix. These labels are updated as read and write operations happen in the system. The access decisions are made by these labels to prevent any implicit write object access, which is unauthorized explicitly. Zimmerman et al. [60] have a similar setup, but with policies that determine any unauthorized write access. Jaume et al. [27] propose a dynamic label updating procedure that detects if there is any flow of data that create a leakage.

CHAPTER 3
PRELIMINARIES

In this section, we cover present definitions and concepts that would be helpful to understand the primary work presented in this study. We give definitions for Discretionary Access Control and Role Based Access Control models, their extensions and the administrative model of RBAC.

Definition 1. (Discretionary Access Control Configuration). A DAC configuration $C = \langle S, O, A \rangle$ is a tuple composed of subjects $s \in S$, objects $o \in O$ and access control rights $A = A_R \cup A_W$ where $A_R \subseteq O \times S$, and $A_W \subseteq S \times O$. We always assume that O and S are disjoint. A pair $(o, s) \in A_R$, also denoted $o \rightarrow_r s$, is a permission representing that subject s can *read* object o. Similarly, a pair $(s, o) \in A_W$, denoted $s \rightarrow_w o$, is a permission representing that subject s can *write* into object o[1]

Definition 2. (Graph Representation of DAC). A DAC can be naturally represented with a bipartite directed graph. The *graph of a DAC* $C = \langle S, O, A \rangle$, denoted $G_C = \langle S \cup O, A \rangle$ is the bipartite graph whose partition has the vertices S and O.

In our study, we provide security analysis for RBAC extensions. Here we give

[1]For the sake of simplicity, we consider only read and write permissions.

definitions to for RBAC, Administrative RBAC and security analysis of RBAC.

Definition 3. (Role Based Access Control Configuration). An RBAC configuration [19] C_R is a tuple $\langle U, R, PRMS, UA, PA, RH \rangle$ where U, R and $PRMS$ are finite sets of users, roles, and permissions, respectively, $UA \subseteq U \times R$ is the user-role assignment relation, $PA \subseteq PRMS \times R$ is the permission-role assignment relation and $RH \subseteq R \times R$ is the role-role assignment (role hierarchy) relation. A pair $(u, r) \in UA$ represents that user u belongs to role r. Similarly, $(p, r) \in PA$ represents that members of role r are granted permission p. A pair $(r_1, r_2) \in RH$ denotes r_1 is superior to r_2, so that any user who has r_1 assigned, also has r_2 assigned, and hence the permissions that are assigned to r_2.

The *Administrative RBAC* (ARBAC) [42] model specifies rules to modify an RBAC configuration. It is composed of three modules `URA` user to role administration, `PRA` permission to role administration, and `RRA` role hierarchy administration.

The `URA` module allows to make changes to UA by using assignment / revocation rules performed by administrators. Administrators are those users that belong to administrative roles. We denote the set of administrative roles as AR. Some policies consider the set AR to be disjoint from the set of roles R. Those policies are said to meet the *separate administration* constraint [47]. A user can be assigned to a role if she satisfies the *precondition* associated to that role. A *precondition* is a conjunction of literals, where each literal is either in positive form r or in negative form $\neg r$, for some role $r \in R$. Following [22], we represent preconditions by two sets of roles Pos and Neg. A user u satisfies a precondition

(Pos, Neg) if u is member of all roles in Pos and does not belong to any role of Neg.

Rules to assign users to roles are specified by the set:

$$\texttt{can\_assign} \subseteq AR \times 2^R \times 2^R \times R.$$

A `can_assign` tuple $(admin, Pos, Neg, r) \in$ `can_assign` allows a member of the administrative role $admin$ to assign a user u to roles r provided that u's current role memberships satisfies the precondition (Pos, Neg).

Rules to revoke users from roles are specified as follows:

$$\texttt{can\_revoke} \subseteq AR \times R.$$

If $(admin, r) \in$ `can_revoke`, a member of the administrative role $admin \in AR$, can revoke the membership of any user from role $r \in R$.

`PRA` is the module to control the permission-role assignments. The rules are similar to those in `URA`. These are defined as follows:

$$\texttt{can\_assignp} \subseteq AR \times 2^R \times 2^R \times R$$

$$\texttt{can\_revokep} \subseteq AR \times R$$

Finally the ARBAC has `RRA` component to perform operations on roles and role hierarchies. The rule defined for this context is the following:

$$\texttt{can\_modify} \subseteq AR \times 2^R$$

Using this rule, authorized administrators can create and remove roles and also they can modify the relationships between the roles.

A URA can be seen as a state-transition system defined by the tuple $\mathcal{S} = \langle U, R, UA, \texttt{can\_assign}, \texttt{can\_revoke}\rangle$. A *configuration* of $\mathcal{S}$ is any user-role assignment relation $UR \subseteq U \times R$. A configuration UR is *initial* if $UR = UA$. Given two $\mathcal{S}$ configurations $c = UR$ and $c' = UR'$, there is a *transition* (or *move*) from c to c' with rule $m \in (\texttt{can\_assign} \cup \texttt{can\_revoke})$, denoted $c \xrightarrow{\tau_m} c'$,if there exists an *administrative* user ad and administrative role $admin$ with $(ad, admin) \in UR$ and a user $u \in U$, and one of the following holds:

can-assign move: $m = (admin, P, N, r)$, $P \subseteq \{r' \mid (u, r') \in UR\}$, $N \subseteq R \setminus \{r' \mid (u, r') \in UR\}$, and $UR' = UR \cup \{(u, r)\}$;

can-revoke move: $m = (admin, r)$, $(u, r) \in UR$, and $UR' = UR \setminus \{(u, r)\}$.

A *run* (or *computation*) of $\mathcal{S}$ is any finite sequence of $\mathcal{S}$ transitions $\pi = c_1 \xrightarrow{\tau_{m_1}} c_2 \xrightarrow{\tau_{m_2}} \ldots c_n \xrightarrow{\tau_{m_n}} c_{n+1}$ for some $n \geq 0$, where c_1 is the *initial* configuration of $\mathcal{S}$. An $\mathcal{S}$ configuration c is *reachable* if c is the last configuration of a run of $\mathcal{S}$.

Definition 4. Reachability Problem (Rights Leakage Problem) for RBAC: Given a URA system $\mathcal{S}$ over the set of roles R and a role $\texttt{goal} \in R$ and a user u, the *role-reachability problem* asks whether a configuration c with $(\texttt{u}, \texttt{goal}) \in \texttt{c}$ is reachable in $\mathcal{S}$.

The reachability problem, seeks to answer certain questions including and not limited to the following [36]:

- **Simple Safety:** Is there a reachable state in which user u belongs to a user set $U_s \subseteq U$? Eventually, this can also be stated as: Can user u ever get access to the roles assigned to users that belong to set U_s?

- **Simple Availability:** In each reachable state, does a user u always belong to a user set $U_s \subseteq U$? Hence this analysis questions whether user u will lose his/her privileges in the future.

- **Bounded Safety:** In each reachable state, is the user set $U_s \subseteq U$ always bounded by $\{u_1, u_2, ..., u_n\}$?

- **Liveness:** In every reachable state, does user set $U_s \subseteq U$ always have at least one user?

- **Containment:** In every reachable state, does a user set U_{s_1} always cover user set U_{s_2}.

Temporal RBAC: The basis of the temporal RBAC models in the literature relies on a *Calendar* definition, which is a periodic and duration expression given in terms of some calendars as follows [10]:

$$P = \sum_{i=1}^{n} O_i \cdot C_i \triangleright r \cdot C_d$$

This expression is composed of two different calendar expressions split by $\triangleright$. The first part is the periodic expression which denotes the starting points of the time intervals represented by the expression. Each $C_i, C_i \sqsubseteq C_{i-1}$ is a calendar represents a different unit time (days, weeks, minutes) so that for each $C_i \sqsubseteq C_{i-1}$ C_{i-1} can be covered by a finite number of intervals of C_i (for instance 24 hours is 1

day). The O_i's are the frequency components associated with the calendars, which are defined as $O_1 = all,\ O_i \in 2^{\mathbb{N}} \cup \{all\}$. The second part of the expression is the duration constraint which describe the time interval that the expression covers once stared with the periodic expression given in the first part. Here, $r = \mathbb{N}$ and $C_d \sqsubseteq C_n$, meaning that the duration cannot exceed the maximum periodic time interval. An example for this expression is that $all.Years + \{3, 7\}.Months \triangleright 2.Months$ means a two month interval that starts every year at the beginning of the third and the seventh months.

The TRBAC model [11] supports role enabling, which is a tuple $\langle r, P \rangle$ composed of roles and calendar expression. In GTRBAC model [33], user to role and permission to role assignments are also proposed to be temporal with the calendar expression.

Previous studies propose temporal role hierarchies [31, 32, 33] that focus on the permission and activation inheritance in the presence of temporal constraints on role enabling and disabling. Particularly, the role hierarchy is still *static*, but the other temporal components of the system have a governing effect on whether the hierarchies will provide inheritance relationship at a given time. These studies propose three different types of hierarchies for temporal domain:

1. *Inheritance-Only Hierarchy* ($\geq$): In this relationship, the permissions in the junior role can be acquired by any user who activated a senior role, without activating the junior role. This hierarchy becomes restricted, if the enabling times of the roles are taken into account.

   There are two types of restrictions possible: Weak and Strong. When a

hierarchy is weakly restricted, then the permission acquisition through the junior role is possible regardless of that role being enabled at that time. However, in the case of strongly restricted hierarchy, the junior role must be enabled to perform permission acquisition.

2. *Activation-Only Hierarchy* ($\succeq$): In this relationship, a user who activated a senior role can activate a junior role even if she is not explicitly assigned to it. This hierarchy becomes restricted, if the enabling times of the roles are taken into account.

   Similar to the Inheritance-Only case, there are two types of restrictions possible: Weak and Strong. When a hierarchy is weakly restricted, then the role activation of the junior role is possible regardless of that role being enabled at that time. However, in the case of strongly restricted hierarchy, the junior role must be enabled in order to be activated through the senior role.

3. *Inheritance and Activation (General Inheritance) Hierarchy* ($\gg$): This relationship is a combination of above two hierarchies. Senior roles can activate junior roles or just inherit some of the permissions of them.

   Lastly, a *Hybrid Hierarchy* exists when the pairwise relations among different roles are of different types. Hence, there can be an inheritance only relation between two roles, and an activation only relation between two other roles in the same hierarchy.

CHAPTER 4
ANALYSIS OF RIGHTS LEAKAGE PROBLEM

The rights leakage problem for RBAC is well-studied and many different security analysis methodologies have been proposed. Hence, in this chapter, our purpose is to study the rights leakage problem for Temporal and Spatio-temporal extensions of RBAC, and propose a security analysis methodology that is scalable to large datasets. The chapter is organized as follows: We first focus on the Temporal RBAC model and its security analysis for rights leakage problem. We provide the methodologies and the experimental results. Later in the chapter, we extend the security analysis of temporal RBAC to the Spatio-temporal RBAC and provide the security analysis experiments for this extension. [1]

4.1 Overview and Challenges of the Rights Leakage Problem in TRBAC

The rights leakage problem in temporal domain requires determining how the *time* is embedded into the model and which components of the model are affected by this. Furthermore, an *administrative model* is necessary to allow certain changes in the these components. Then, a security analysis is possible for the TRBAC model. In this section, we first discuss on how temporal constraints can be applied to user-role assignment, permission-role assignment, role enabling and role-role

[1] The contributions in this chapter were published in [50, 51, 52]

assignment (role hierarchy) relations. Next, we present the administrative model that governs modifications to these relations which leads us to our proposed security analysis for TRBAC model.

In the RBAC models with temporal components proposed earlier in the literature, the majority of them focus extensively on benefits of having temporal constraints on the temporal user-role assignment relation and role enabling. In this study, we do not limit ourselves to cover only these two relations, and focus on the two other relations, namely, permission-role assignment and role hierarchies, as well. Since the benefits of having temporal user-role assignment and role enabling have been discussed extensively by earlier studies, we now discuss potential benefits of having temporal permission-role assignments and temporal role hierarchies.

Temporal permission-role assignments capture the changes in PA with respect to time, hence, a role can have different permission assignments in different time intervals. This concept, although look similar to temporal UA, can have different applications in a TRBAC model, including reducing the number of roles. Let us explain this with an example:

Example 1. Consider a manufacturing company has two different production plants in different cities, one also has the headquarters of the company. The company has a CEO and a General Manager (GM) who works at both the plants; an Accounting Manager (AM), a Manufacturing Manager (MM), and a Human Resources Manager (HR) for each plant. Although the CEO works at the headquarters, GM works in both of the plants in different days of the week. When he is present at a plant, he manages the operations and audits the actions of the AM

of that plant. However, when he is away (at the other plant), MM has the responsibility to audit the operations of AM without completely assuming the GM role, which is considered to have many additional permissions. In this case a TRBAC model without Temporal PA must have two different roles for each MM: Regular MM and Extended MM, and in Temporal UA the necessary assignments are done. However, presence of Temporal PA allows the model to have only one MM role that has different permission assignments that captures the auditing process whenever necessary.

In the Temporal RBAC model, role hierarchies can also be temporal in nature, in other words, they may change with respect to temporal constraints. Although role hierarchies in prior temporal extensions of RBAC have been specified, they do not allow temporal constraints to be specified on them that not only *restrict the time* during which the hierarchy is valid, but also *change its structure* by shifting the position of the roles in the hierarchy. An immediate effect of this is that permission inheritance does not always hold. Essentially this means that a senior level role cannot always inherit the permissions of a junior level role. Furthermore, a role may change its level in the hierarchy, for example, a junior level role may be elevated to a higher level role during certain time periods.

Although enterprises usually specify a static hierarchy, a *dynamic temporal role hierarchy* (DTRH) comes into play in some temporary or periodical exceptional situations that are required for operational purposes. In the following, we provide such a motivating example.

Example 2. Consider once again the manufacturing company given in the previ-

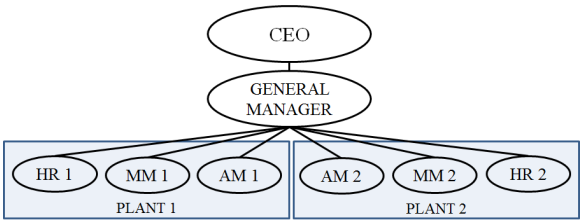


Figure 4.1. The Role Hierarchy of the Manufacturing Company

ous example. The auditing tasks of MM can be modeled with DTRH, if the tasks required for auditing can be acquired through the role hierarchy given in Figure 4.1. A policy which makes the Manufacturing Manager move to the second level, on top of the Accounting Manager only on the days when the General Manager is away will provide permissions needed for auditing the AM to MM.

Nevertheless, it is still possible to represent the scenario in the example above using a static role hierarchy. However, lack of temporal role hierarchies will force the system administrators to create a dummy role, like “Manager and Auditor”, that is only enabled when necessary. Also, this role should have the required permission and hierarchy assignments that Manufacturing Manager needs. This newly created role does not essentially represent a regular job function since the Manufacturing Manager cannot assume this role all the time. Moreover, the Manufacturing Manager should be assigned to two separate roles which are enabled and disabled in regular time intervals. The situation might get even more complicated in the case of temporary changes in the system. Suppose that this auditing position is applied only when the General Manager is on vacation. Then the newly created dummy role and the necessary permission assignments are performed just for a single and temporary occurrence. Even worse, the administrators must undo the changes in the system, by revoking and deleting this role when the

General Manager returns. Skipping this step would create serious safety problems. Clearly, creation of these redundant dummy roles increases the administrative burden [25].

Role delegation, which has been studied extensively in the literature [1, 7, 8, 17, 30, 54, 58, 59], is another way of handling scenarios like this. Users are delegated to the necessary roles of the users that are away. Even though our example scenario can be modeled using role delegation without imposing significant overhead, using temporal role hierarchies has still an advantage in terms of performing security analysis. Whether handling the temporal role hierarchies is done using the specification of DTRH, using dummy roles or delegation, none of the prior work on safety analysis considers RBAC models with temporal constraints on role hierarchies.

4.2 TRBAC and its Administrative Model

Although Temporal RBAC models have been proposed in the literature, none of them addresses the security analysis of policies. The temporal dimension of the model makes it even harder to perform security analysis, which is already proved to be intractable for the non-temporal case. Therefore, while preserving the core idea of having the temporal notion embedded into the RBAC components as in [11, 33], we simplify the model to allow for a manageable security analysis. Although, our simplified model does not completely represent the previous temporal models, such as TRBAC or GTRBAC as a whole, we call this model Temporal RBAC (or TRBAC, in short) for notational simplicity. Therefore, the model referred as TRBAC for the remainder of this chapter represents our simplified

model, unless otherwise noted. Now, we explain our TRBAC model in detail. We first define how the *time* is represented in the model:

Let T_{MAX} be a positive integer. A *time slot* of $Times$ is a pair $(a, a+1)$, where a is an integer, and $0 \leq a < a+1 \leq T_{MAX}$. A time slot $(a, a+1)$ represents the set of all times in the set $[a, b)$, i.e., $\{t \mid a \leq t < b\}$. We use a *time interval*, consisting of a pair (a, b) where a, b are two integers and $0 \leq a < b \leq T_{MAX}$, to represent the set of corresponding time slots $\{(a, a+1), (a+1, a+2), \ldots (b-1, b)\}$ succinctly. A *schedule* over T_{MAX} is a set of time slots.

For instance, consider a hospital that works for 24 hours with three shifts (between 9 am and 5 pm, between 5 pm and 1 am, and between 1 am and 9 am). If we want to have the precision of hours, we choose $T_{MAX} = 24$, and a schedule s that covers shifts 9 am–5 pm and 5 pm–1 am is represented as $s = \{(9, 10), (10, 11) \ldots, (23, 24), (24, 1)\}$. The schedule definition is a simplified version of the Calendar definition in Bertino et al. [11], where we have simpler periodic constraints and do not have duration constraints.

We assume that the system is periodic, thus the schedules repeat themselves after any T_{MAX}; in the hospital example above, time intervals are repeated each 24 hours. Given a schedule s over T_{MAX} and an real number t, we say that t belongs to s, denoted $t \in s$, if there is a time interval $(a, b) \in s$ such that $t' \in [a, b)$, where $t' = t \; mod \; T_{MAX}$.

Definition 5. (TRBAC Configuration). Let S be the set of all possible schedules over T_{MAX}. A TRBAC configuration over T_{MAX} is a tuple $M = \langle U, R, PRMS,$ $TUA, TPA,$

$RS, DTRH\rangle$ where U, R and $PRMS$ are finite sets of *users*, *roles*, and *permissions*, respectively, $TUA \subseteq (U \times R \times S)$ is the *temporal user to role assignment* relation, $TPA \subseteq (PRMS \times R \times S)$ is the *temporal permission to role assignment* relation, $RS \subseteq (R \times S)$ is the *role-status* relation and $DTRH$ is the *dynamic temporal role hierarchy* relation.

A tuple $(u, r, s) \in TUA$ represents that user u is a member of the role r only during the time intervals of schedule s. During the life time of the system, a role can be either enabled or disabled. A tuple $(r, s) \in RS$ imposes that role r is *enabled* only during the time intervals of s (and therefore it can be assumed to be a member of r only at these times), and *disabled* otherwise. A tuple $(p, r, s) \in TPA$ means that permission p is associated to role r only in the time intervals denoted by s. Thus, a user u is granted permission p at time $t \in [0, T_{MAX}]$ provided that there exists a role $r \in R$ such that $(u, r, s_1) \in TUA$, $(r, s_2) \in RS$, $(p, r, s_3) \in PA$, and $t \in (s_1 \cap s_2 \cap s_3)$, for some time intervals s_1, s_2 and s_3.

We assume that relation RS for each role $r \in R$ contains always exactly one pair with first component r. Similarly, the relation TUA contains exactly one tuple for each pair in $U \times R$. Thus, if a role r is disabled in any time interval, we require that RS relates r with the empty schedule. Similarly, if a user u does not belong to a role r in any time interval, the pair (u, r) is associated to the empty schedule by the relation TUA.

Permission inheritance and role activation through role hierarchies require additional definitions. In our model, $DTRH$ is represented as a collection of dynamic temporal role hierarchy policies, which are tuples consisted of a pair of

roles associated with a schedule that denotes the time slots that the policy is valid. In our model, we have dynamic temporal role hierarchy for inheritance only relation $DTRH_I$, for activation only relation $DTRH_A$ and for general inheritance relation $DTRH_{IA}$, all comprises as $DTRH = DTRH_I \cup DTRH_A \cup DTRH_{IA}$.

Definition 6. A *dynamic temporal role hierarchy policy* $(r_1 \geq_{s,weak} r_2) \in DTRH_I$ between roles r_1 and r_2 is an *inheritance-only weak temporal hierarchy relation*, that is valid in the time slots specified by a schedule s. Under this policy, a user u who can activate r_1 can inherit permissions of r_2 at time t if (1) $(u, r_1, s_1) \in TUA$ (2) $(r_1, s_2) \in RS$ and (3) $t \in (s_1 \cap s_2 \cap s)$, provided that there exists schedules s_1 and s_2 that determine the time slots that u is assigned to r_1 and r_1 is enabled, respectively.

Definition 7. A *dynamic temporal role hierarchy policy* $(r_1 \succeq_{s,weak} r_2) \in DTRH_A$ between roles r_1 and r_2 is an *activation-only weak temporal hierarchy relation*, that is valid in the time slots specified by a schedule s. Under this policy, a user u can activate r_2 at time t if (1) $(u, r_1, s_1) \in TUA$ (2) $(r_2, s_2) \in RS$ and (3) $t \in (s_1 \cap s_2 \cap s)$, provided that there exists schedules s_1 and s_2 that determine the time slots that u is assigned to r_1, and r_2 is enabled, respectively.

Definition 8. A *dynamic temporal role hierarchy policy* $(r_1 \gg_{s,weak} r_2) \in DTRH_{IA}$ between roles r_1 and r_2 is a *general weak temporal hierarchy relation*, that is valid in the time slots specified by a schedule s. Under this policy, a user u can activate r_2 at time t, or inherit permissions of r_2 if (1) $(u, r_1, s_1) \in TUA$ (2) $(r_2, s_2) \in RS$ and (3) $t \in (s_1 \cap s_2 \cap s)$, provided that there exists schedules s_1, and s_2 that determine the time slots that u is assigned to r_1 and r_2 is enabled, respectively.

In the above three definitions, the relations become strong, (i.e: $r_1 \geq_{s,strong} r_2) \in DTRH_I$, $(r_1 \succeq_{s,strong} r_2) \in DTRH_A$ and $(r_1 \gg_{s,strong} r_2) \in DTRH_{IA}$), when (2) is replaced with $(r_1, s_2), (r_2, s_3) \in RS$ and (3) is replaced with $t \in (s_1 \cap s_2 \cap s_3 \cap s)$ where s_3 is the schedule that determine the time slots that r_2 is enabled.

Presence of more than one type of relation makes $DTRH$ a *hybrid* hierarchy.

Dynamic temporal role hierarchy policies $(r_1 \geq_{s,weak} r_2) \in DTRH$ satisfy the following properties for a given schedule s:

1. *Reflexive:* $(r_1 \geq_{s,weak} r_1) \in DTRH$
2. *Transitive:* If $(r_1 \geq_{s,weak} r_2), (r_2 \geq_{s,weak} r_3) \in DTRH$, then $(r_1 \geq_{s,weak} r_3) \in DTRH$.
3. *Asymmetric:* If $(r_1 \geq_{s,weak} r_2) \in DTRH$ then $(r_2 \geq_{s,weak} r_1) \notin DTRH$. These properties apply for both strong and the other types of relations ($\succeq$, $\gg$) as well.

Different policies among different roles create derived relations. As discussed in [32] derived relations determine the scope of activation or inheritance privileges upon activating a role. We adopt these derived relations to the case of dynamic temporal role hierarchies as follows:

Definition 9. A **derived relation** among roles $x, y_1, y_2, ..., y_n, z \in R$ holds under any of the following conditions:

1. $(x\langle\mathcal{F}\rangle_{s_0,type} y_1) \wedge (y_1\langle\mathcal{F}\rangle_{s_1,type} y_2) \wedge ... \wedge (y_{n-1}\langle\mathcal{F}\rangle_{s_{n-1},type} y_n) \wedge (y_n\langle\mathcal{F}\rangle_{s_n,type} z) \rightarrow (x\langle\mathcal{F}\rangle_{s,type} z)$ if $\mathcal{F} \in \{\geq, \succeq, \gg\} \wedge s = s_0 \cap ... \cap s_n$,

2. $(x \geq_{s_0,type} y_1) \wedge (y_1 \langle \mathcal{F} \rangle_{s_1,type} y_2) \wedge ... \wedge (y_{n-1} \langle \mathcal{F} \rangle_{s_{n-1},type} y_n) \wedge (y_n \langle \mathcal{F} \rangle_{s_n,type} z) \rightarrow (x \geq_{s,type} z)$ if $\mathcal{F} \in \{\geq, \gg\} \wedge s = s_0 \cap ... \cap s_n$,

3. $(x \gg_{s_0,type} y_1) \wedge (y_1 \langle \mathcal{F} \rangle_{s_1,type} y_2) \wedge ... \wedge (y_{n-1} \langle \mathcal{F} \rangle_{s_{n-1},type} y_n) \wedge (y_n \langle \mathcal{F} \rangle_{s_n,type} z) \rightarrow (x \langle \mathcal{F} \rangle_{s,type} z)$ if $\mathcal{F} \in \{\geq, \gg\} \wedge s = s_0 \cap ... \cap s_n$,

4. $(x \gg_{s_0,type} y_1) \wedge (y_1 \succeq_{s_1,type} y_2) \wedge ... \wedge (y_{n-1} \succeq_{s_{n-1},type} y_n) \wedge (y_n \succeq_{s_n,type} z) \rightarrow (x \succeq_{s,type} z)$ if $s = s_0 \cap ... \cap s_n$.

The other rules stated in [32] hold as in the above definition provided that $s = (s_1 \cap s_2 \cap ... \cap s_n) \neq \emptyset$.

According to Definition 9, if all of the linked hierarchy policies are of same type, the derived policy is also of the same type. If the first policy is an inheritance only relation, then regardless of the other linked policies being activation only or general inheritance hierarchy, the derived relation will be an inheritance-only policy. Similarly, if the first policy is a general inheritance relation and the remaining policies are activation-only, the derived relation is an activation-only policy. Finally, if the first policy is a general inheritance relation and the other linked policies being activation only or general inheritance relations, the derived relation will be of the type of linked policies.

Now, we can present our administrative model that allows administrators to make changes to the role-status relation RS,temporal user to role assignment relation TUA, temporal permission to role assignment relation TPA and the dynamic temporal role hierarchy relation $DTRH$ by using enable / disable, assignment / revocation and modify rules, respectively. The goal of these rules is to update the time intervals of the schedule s associated to the corresponding relation.

In the analysis of the TRBAC model, we assume that the analysis for TPA can be made separately, since it is not directly related to the analysis of other components in terms of the security questions in consideration. More specifically, the security questions ask whether it is possible for a user to get access to a role, which requires determining whether it is possible for the goal role to be assigned to the target user directly, or indirectly via the role hierarchy in a time interval and if the role is enabled during any portion of this time interval. On the other hand, the analysis for TPA is needed to discover if there is a possibility for a permission to appear in a particular goal role. Therefore, we define the Temporal URA and Temporal PRA systems separately to observe the state transitions.

Definition 10. (Temporal User to Role Administration). A TURA system is a tuple $\mathcal{S}_{\scriptscriptstyle T} = \langle M, \texttt{can\_enable}, \texttt{can\_disable}, \texttt{t\_can\_assign}, \texttt{t\_can\_revoke}, \texttt{t\_can\_modify}\rangle$ where $M = \langle U, R, PRMS, TUA, TPA, RS, DTRH\rangle$ is a TRBAC policy over T_{MAX}, and $\texttt{can\_enable}$, $\texttt{can\_disable}$, $\texttt{t\_can\_assign}$, $\texttt{t\_can\_revoke} \subseteq (R \times S \times 2^R \times 2^R \times S \times R)$ and $\texttt{t\_can\_modify} \subseteq (R \times S \times 2^R \times 2^R \times 2^R \times 2^R \times S \times R \times R \times \{strong, weak\} \times \{\geq, \succeq, \gg\})$.

Definition 11. (Temporal Permission to Role Administration). A TPRA system is a tuple $\mathcal{S}_{\scriptscriptstyle T} = \langle M, \texttt{t\_can\_assignp}, \texttt{t\_can\_revokep}\rangle$ where $M = \langle U, R, PRMS, TUA, TPA, RS, DTRH\rangle$ is a TRBAC policy over T_{MAX}, and $\texttt{t\_can\_assignp}$, $\texttt{t\_can\_revokep} \subseteq (R \times S \times 2^R \times 2^R \times S \times R)$.

A *configuration* of $\mathcal{S}_{\scriptscriptstyle T}$ for TURA is a triple $(RS, TUA, DTRH)$, which is *initial* if $RS = RS_0$, $TUA = TUA_0$ and $DTRH = DTRH_0$. Similarly, a *configuration* of $\mathcal{S}_{\scriptscriptstyle T}$ for TPRA is a singleton (TPA), which is *initial* if $TPA = TPA_0$.

Given two $\mathcal{S}_T$ configurations $c = (RS, TUA, DTRH)$ and $c' = (RS', TUA', DTRH')$ for TURA' and $c = (TPA)$ and $c' = (TPA')$ for TPRA, we describe below the conditions under which there is a *transition* (or *move*) from c to c' at time $t \in \mathbb{N}$ with rule $m \in \mathcal{M}_{ALL} = (\texttt{can\_enable} \cup \texttt{can\_disable} \cup \texttt{t\_can\_assign} \cup \texttt{t\_can\_revoke} \cup \texttt{t\_can\_modify} \cup \texttt{t\_can\_assignp} \cup \texttt{t\_can\_revokep})$, denoted $c \xrightarrow{(\tau_m, t)} c'$.

Before defining the transition relation, we first describe the components of move $m = (admin, s_{rule}, Pos, Neg, s_{role}, r)$. Move m can be executed only by a user, say ad, belonging to the *administrative role* $admin \in R$.

The times t in which ad can execute m are all those in which ad is assumed to be a member of role $admin$, and furthermore, t must also belong to the schedule s_{rule} which denotes the time intervals when m can be fired (or we say *valid*): $t \in (s_{ad} \cap s_{admin} \cap s_{rule})$ where $(ad, admin, s_{ad}) \in TUA$ and $(admin, s_{admin}) \in RS$. In the rest of the section we say that m can be *executed* at time t whenever t fulfills the above condition. The component s_{role} is used to update the schedule of a role, or the membership of a user to a role, depending on the kind of rule of m. The pair of disjoint role sets (Pos, Neg) is called the *precondition* of m whose fulfillment depends by the kind of the rule m.

The fulfillment of the precondition of a can-enable and can-disable rule depends on the current status of the other roles. Let $\hat{s} \subseteq s_{role}$. A can-enable or can-disable rule $m = (admin, s_{rule}, Pos, Neg, s_{role}, r)$ satisfies its precondition (Pos, Neg) w.r.t. candidate schedule $\hat{s}$, if for every time slot $\alpha \in \hat{s}$, if (1) for every role $pos \in Pos$, $\alpha \subseteq s_{pos}$ where $(pos, s_{pos}) \in RS$, (2) for every role $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, where $(neg, s_{neg}) \in RS$, and (3) α satisfies all preconditions. In

other words, a candidate schedule $\hat{s} \subseteq s_{role}$ satisfies a precondition only if each time slot $\alpha \in \hat{s}$ satisfies the precondition individually. Let $(r, s) \in RS$.

Enabling Rules: A can-enable rule adds a new schedule to a specific role. A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `can_enable` allows to update the tuple $(r, s) \in RS$ to $(r, s \cup \hat{s})$ for some schedule $\hat{s}$, provided that m can be executed at time t and also satisfies its precondition. Formally, rule m is executable at time t, m satisfies its precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, $RS' = (RS \setminus \{(r, s)\}) \cup \{(r, s \cup \hat{s})\}$, and $TUA' = TUA$.

Disabling Rules: A can-disable rule removes a schedule from a designed role. A tuple $m = (admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `can_disable` allows to update the tuple $(r, s) \in RS$ to $(r, s \setminus \hat{s})$, for some schedule $\hat{s}$, provided that m can be executed at time t, and satisfies its precondition. Formally, m is executable at time t, m satisfies its precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, $RS' = (RS \setminus \{(r, s)\}) \cup \{(r, s \setminus \hat{s})\}$, and $TUA' = TUA$.

The next two rules are similar to those given above with the difference that we now update the schedules associated to each element of the user to role assignment relation. Another difference is that can-assign and can-revoke rules have a different semantics to fulfill their preconditions. A user $u \in U$ satisfies a precondition (Pos, Neg) w.r.t. a schedule $\hat{s}$ if for every time slot $\alpha \in \hat{s}$, (1) for every $(u, pos, s_{pos}) \in TUR$ with $pos \in Pos$, $\alpha \subseteq s_{pos}$, (2) for every $(u, neg, s_{neg}) \in TUA$ with $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, and (3) α satisfies all preconditions. Let $(u, r, s) \in TUA$.

Assignment Rules: A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_assign` allows to update the user to role assignment relation for the pair (u, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and user u satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (u, r, s) is updated to $(u, r, s \cup \hat{s})$, i.e. $TUA' = (TUA \setminus \{(u, r, s)\}) \cup \{(u, r, s \cup \hat{s})\}$, and $RS' = RS$.

Revocation Rules: A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_revoke` allows to update the user to role assignment relation for the pair (u, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and user u satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (u, r, s) is updated to $(u, r, s \setminus \hat{s})$, i.e. $TUA' = (TUA \setminus \{(u, r, s)\}) \cup \{(u, r, s \setminus \hat{s})\}$, and $RS' = RS$.

The rules for updating the permission to role assignment is again similar to the user to role assignments rules, with the difference of assigning permissions and preconditions checked against the assigned permissions. The structure of the move definition is similar to the existing model, but the assignment semantics for permissions are different. Hence, the existing move definition, $m = (admin, s_{rule}, Pos, Neg, s_{role}, r)$ remains the same, but it applies to permissions rather than users.

Intuitively, a precondition in the permission level is a verification procedure of the existing role assignments of a given permission. For instance, a positive precondition (negative, resp.) can state a permission can only be added to a given role if it has already been (has not been, resp.) assigned to another role. More formally, a permission $p \in PRMS$ satisfies a precondition (Pos, Neg) w.r.t. a

schedule $\hat{s}$ if for every time slot $\alpha \in \hat{s}$, (1) for every $(p, pos, s_{pos}) \in TPA$ with $pos \in Pos$, $\alpha \subseteq s_{pos}$, (2) for every $(p, neg, s_{neg}) \in TPA$ with $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, and (3) α satisfies all preconditions. Let $(p, r, s) \in TPA$.

Assignment Rules: A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_assignp` allows to update the permission to role assignment relation for the pair (p, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and permission p satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (p, r, s) is updated to $(p, r, s \cup \hat{s})$, i.e. $TPA' = (TPA \setminus \{(p, r, s)\}) \cup \{(p, r, s \cup \hat{s})\}$, $TUR' = TUR$ and $ER' = ER$.

Revocation Rules: A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_revokep` allows to update the permission to role assignment relation for the pair (p, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and permission p satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (p, r, s) is updated to $(p, r, s \setminus \hat{s})$, i.e. $TPA' = (TPA \setminus \{(p, r, s)\}) \cup \{(p, r, s \setminus \hat{s})\}$, $TUR' = TUR$ and $ER' = ER$.

The rule structure for `t_can_modify` is different from the other rules. This rule updates the valid time slots of the dynamic temporal role hierarchy policies. Also, in contrast to precondition structures that have been proposed in the literature for other administrative rules (like `can_assign`), it has two sets of preconditions, one for senior and one for junior role in order to protect the integrity of the hierarchy. The rule is composed of eight parameters that should be satisfied to execute the rule. Three of these parameters are similar to the above mentioned moves, namely, $admin$, s_{rule} and $s_{hierarchy}$ which is declared as s_{role} in other rules defined above, but has similar semantics. Let t be the time slot that the rule is required to be

executed.

- $type \in \{strong, weak\}$ denotes the type of the hierarchy relation.
- r_{sr} is the senior role of the hierarchy policy.
- r_{jr} is the junior role of the hierarchy policy.
- $SR(Pos, Neg)$ denotes the positive and negative preconditions of the senior role r_{sr}. The preconditions are satisfied in the following way: Let $\hat{s}$ denote the time slots that are intended to be modified by the rule ($\hat{s} \subseteq s_{hierarchy}$). For each $r \in Pos$, there must be a role hierarchy policy $(r \geq_{\hat{s},type} r_{sr}) \in DTRH$ and for each $r \in Neg$, there must not be a hierarchy policy $(r \geq_{\hat{s},type} r_{sr}) \in DTRH$.
- $JR(Pos, Neg)$ denotes the positive and negative preconditions of the junior role r_{jr}. The preconditions are satisfied in the following way. Let $\hat{s}$ denote the time slots that are intended to be modified by the rule ($\hat{s} \subseteq s_{hierarchy}$). For each $r \in Pos$, there must be a role hierarchy policy $(r_{jr} \geq_{\hat{s},type} r) \in DTRH$ and for each $r \in Neg$, there must not be a hierarchy policy $(r_{jr} \geq_{\hat{s},type} r) \in DTRH$.

Modification Rule: Under these parameters, a tuple: $(admin, s_{rule}, SR(Pos, Neg), JR(Pos, Neg), s_{hierarchy}, r_{sr}, r_{jr}, type) \in$ `t_can_modify` allows to update the role hierarchy relation $r_{sr} \geq_{s,type} r_{jr}$ as follows: Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{hierarchy}$. Then, if this rule can be executed at time t, and the preconditions are satisfied w.r.t. schedule $\hat{s}$, then the tuple $r_{sr} \geq_{s,type} r_{jr}$ is updated to

$r_{sr} \geq_{s \cup \hat{s}, type} r_{jr}$ or $r_{sr} \geq_{s \setminus \hat{s}, type} r_{jr}$, depending on the intended modification. This definition is for inheritance only hierarchies, but it also applies to activation only and general inheritance hierarchies, by replacing $\geq$ with $\succeq$ and $\gg$.

Example 3. Let us now consider an example of a TRBAC system deployed in a hospital. Assume that there are 7 different roles, namely, Employee (EMP), Day Doctor (DDR), Night Doctor (NDR), Practitioner (PRC), Nurse (NRS), Secretary (SEC) and Chairman (CHR). Hospital works for 24 hours and there are three different shifts (time slots) from 8 am to 4 pm (Time Slot 1), 4 pm to 12 am (Time Slot 2) and 12 am to 8 am (Time Slot 3). Only the Chairman role (CHR) has administrative privileges.

1. (CHR, $\{(0,2)\}$, $\{DDR\}$, $\emptyset$, $\{(0,1)\}$, PRC)$\in$ `can_enable`: At time slots 1 and 2, a chairman can enable the role *Practitioner* for the first time slot if the role *Day Doctor* is also enabled during this time slot.
2. (CHR, $\{(0,3)\}$, $\{EMP, NDR\}$, $\{(2,3)\}$, NRS) $\in$ `can_disable`: At time slots 1, 2 and 3, a chairman can disable the role *Nurse* for the third time slot if the roles *Employee* and *Night Doctor* are enabled at this time slot.
3. (CHR, $\{(0,2)\}$, $\{EMP\}$, $\{NRS\}$, $\{(0,2)\}$, DDR) $\in$ `t_can_assign`: At time slots 1 and 2, a chairman can assign the role *Day Doctor* for the first and the second time slots to any user that has *Employee* role and does not have *Nurse* role during these time slots.
4. (CHR, $\{(0,3)\}$, $\emptyset, \emptyset$, $\{(0,3)\}$, SEC) $\in$ `t_can_revoke`: At time slots 1, 2 and 3,a chairman can revoke the role *Secretary* for all time slots of any user that has *Secretary* role assigned in these slots.

5. (CHR, $\{(2,3)\}$, $\{EMP\}$, $\{NRS\}$, $\{(2,3)\}$, NDR) $\in$ `t_can_assignp`: At time slot 3, a chairman can assign a permission to the role *Night Doctor* for the third time slot if that permission is also assigned to *Employee* not assigned to *Nurse* role during this time slot.

6. (CHR, $\{(0,2)\}$, $\emptyset$, $\emptyset$, $\{(0,3)\}$, NRS) $\in$ `t_can_revokep`: At time slots 1 and 2, a chairman can revoke a permission from the role *Nurse* for all time slots.

7. (CHR, $\{(0,2)\}$, $\{DDR\}$, $\emptyset$, $\{(0,1)\}$, PRC)$\in$ `t_can_assign`: At time slots 1 and 2, a chairman can assign the role *Practitioner* for the first time slot of any user that has *Day Doctor* role during this time slot.

8. (CHR, $\{(0,3)\}$,$\{NDR\}$, $\emptyset$, $\{(2,3)\}$, PRC)$\in$ `t_can_assign`: At time slots 1, 2 and 3, a chairman can assign the role *Practitioner* for the third time slot to any user that has *Night Doctor* role during this time slot.

Reachability problems: A *run* (or *computation*) of $\mathcal{S}_T$ is any finite sequence of $\mathcal{S}_T$ transitions $\pi = c_1 \xrightarrow{(\tau_{m_1}, t_1)} c_2 \xrightarrow{(\tau_{m_2}, t_2)} \ldots c_n \xrightarrow{(\tau_{m_n}, t_n)} c_{n+1}$ for some $n \geq 0$, where c_1 is an *initial* configuration of $\mathcal{S}_T$, $t_1 = 0$, and $t_i \leq t_{i+1}$ for every $i \in [n-1]$. An $\mathcal{S}_T$ configuration c is *reachable within time* t, if there exists a run π in which $c_{n+1} = c$ and $t_n \leq t$. Furthermore, c is simply *reachable* if c is reachable within time t, for some $t \geq 0$.

Let $\mathcal{S}_T$ be a `TURA` system over T_{MAX}, u and r be a user and a role of $\mathcal{S}_T$, respectively, and s be a schedule over T_{MAX}. Given a time t, the *timed reachability problem* for $(\mathcal{S}_T, u, r, s, t)$ asks whether there is a reachable configuration within time t of $\mathcal{S}_T$ in which user u is a member of role r in the schedule s either explicitly

or implicitly through the role hierarchy. Similarly, the *reachability problem* for $(\mathcal{S}_T, u, r, s)$ is defined as above where there is no constraint on time t. In all of the time slots of s, r must also be enabled.

For a TPRA system over T_{MAX} which is identified by $\mathcal{S}_T$, and p and r are a permission and a role of $\mathcal{S}_T$, respectively, and s be a schedule over T_{MAX}. Given a time t, the *timed reachability problem* for $(\mathcal{S}_T, p, r, s, t)$ asks whether there is a reachable configuration within time t of $\mathcal{S}_T$ in which user u is a member of role r in the schedule s. Similarly, the *reachability problem* for $(\mathcal{S}_T, u, r, s)$ is defined as above where there is no constraint on time t.

In our analysis, we assume Separate Administration, in which there is an administrative user who is assigned to the required administrative roles which are enabled all the time. Hence, the times to fire a rule is only restricted by s_{rule}.

Given an initial configuration c_0, rules of an administrative model, $\mathcal{M}_{ALL}$ and the target user u, who is being analyzed against the security questions of interest, our proposed security analysis methodology provides answers to various security questions outlined in Section 4.3.

4.3 TRBAC Security Analysis

In Temporal RBAC, the security problem is slightly different than that of RBAC. The model can have two different ranges of temporal coverage: Safety until a given period of time (or called short term safety), and the ultimate safety (or called long term safety). In short term safety, we are only interested in the safety of the system until a given fixed time. Practically, this type of an analysis is useful to

track users that will have temporary presence in the system. Whereas, the long term safety is more concerned about the regular users which are likely to be active in the system for relatively longer periods of time. This analysis will yield an ultimate safety of the system in the long run. Furthermore, changes allowed in the role hierarchy require additional security questions related to implicit assignments that are possible in the future. There is no problem of this sort in the case of static role hierarchies, however a simple manipulation in the hierarchy could create a security breach, and should be detected in advance to prevent any such occurrence. Considering these definitions, some example security questions for the temporal domain can be stated as follows:

1. **Safety:**

   (a) *(Explicit Assignment - Short Term)* Will there be no reachable state in which a user u is assigned to a role r at time t?

   (b) *(Explicit Assignment - Long Term)* Will a user u ever get assigned to a role r?

   (c) *(Role Enabling - Long Term)* Will an enabled role r eventually be disabled?

   (d) *(Implicit Assignment - Short Term)* Will a user u get *implicitly* assigned to role r at time t?

   (e) *(Implicit Assignment - Long Term)* Will a user u ever get *implicitly* assigned to role r in the future?

   (f) *(Permission Assignment - Long Term)* Will a permission p ever get assigned to role r in the future?

2. **Liveness:**

   (a) *(Role Enabling - Short Term)* Will an enabled role remain enabled at time t?

   (b) *(Implicit Assignment - Short Term)* Will a user u lose privileges of any role that he is *implicitly* assigned until time t?

   (c) *(Explicit Assignment - Long Term)* Will a user u ever lose any role that he is assigned in the future?

   (d) *(Permission Assignment - Short Term)* Will a permission p remain assigned to role r at time t?

3. **Mutual Exclusion:**

   (a) *(Explicit Assignment - Long Term)* Will a user u be assigned to roles r_1 and r_2 at the same time (i.e., do the time intervals during which u is assigned to roles r_1 and r_2 overlap?

   (b) *(Implicit Assignment - Short Term)* Will users u_1 and u_2 get *implicitly* assigned to role r at the same time slot until time t?

Regarding these security questions, our aim is to analyze TRBAC model to verify that the configuration is safe in terms of the questions stated above.

Our security analysis depends on a customizable three stage decomposition strategy. First we decompose the problem into four steps based on the temporal relation that is modified (TUA,RS,TPA,$DTRH$). Then, we further decompose each of these subproblems into smaller ones using the time dimension in which we have two different strategies to address different security questions – Rule

Schedule and Role Schedule. Finally, combining the results obtained from each of these decomposed problems provide the complete analysis.

4.3.1 Stage 1: Relation Based Decomposition

The TURA and TPRA systems are composed of a set of different type of rules that are used to generate new configurations for a security analysis. The interactions among these rules, however, have certain properties. Consider the rules grouped according to the relations that they modify, i.e, t_can_assign, t_can_revoke; can_enable, can_disable; t_can_assignp, t_can_revokep; and t_can_modify are the four groups of rules that modify different relations in TRBAC. Assuming that the administrator role and rule schedule requirements are satisfied, the execution of roles of each group is determined by the relations that they modify. For instance, the preconditions to satisfy for t_can_assign and t_can_revoke are checked against the current status of TUA, whereas, it is TPA for t_can_assignp, t_can_revokep, RS for can_enable, can_disable and $DTRH$ for t_can_modify. Therefore, the execution rules of different groups are independent of each other. However, this property does not imply that the *relations that are modified* with these rules are also independent semantically. For instance, role assignments and revocations can be performed for a user, but these assignments are useful only if the roles are enabled. Similarly, an inheritance through the role hierarchy is only possible if the senior role of the policy is enabled. Therefore, we perform independent analysis on four different components of the TRBAC model and then we combine the results obtained from each of these four sub-analysis problems in order to interpret them correctly in Stage 3.

Hence, regarding this property, our security analysis procedure is composed of four steps (Table 4.1). In each of these steps, the state configurations and the administrative rule sets of the analysis problems are shaped with different relations.

Table 4.1. Subproblems, Initial Configurations and the Relations Used

Analysis Performed	State Configuration Represented By	Initial Configuration of the Analysis
1. User Assignment 2. Role Enabling	TUA RS	TUA Relation of the target user RS Relations of all roles
✓Explicit Role Assignment Analysis is complete.		
3. Role Hierarchy	DTRH	DTRH policies
✓Implicit Role Assignment Analysis is complete.		
4. Permission Assignment	TPA	TPA Relation of the target role
✓Full Analysis is complete.		

For each different analysis, the rule set is composed of the following rules:

1. User Assignment: `t_can_assign`, `t_can_revoke`
2. Role Enabling: `can_enable`, `can_disable`
3. Role Hierarchy: `t_can_modify`
4. Permission Assignment: `t_can_assignp`, `t_can_revokep`

This four step procedure depicted in Table 4.1 might be customized with respect to the scope of the security analysis. At the end of first step, the analysis generates all possible configurations for the target user under the administrative rules. The second step declares the time slots that the roles can get enabled. Combining the results of the first and the second step produces the analysis that answers the security questions related to explicit role assignments. If the implicit assignments

are also considered, the third step should be performed. In the third step, possible role hierarchy relations are generated. Combining these results with the ones from the earlier steps will determine the possibility of an implicit assignment to a role. Finally, the fourth step determines the possible permission assignments to a role (or roles), which could also be conducted as an independent analysis determining whether there is a possibility for a set of permissions to appear in a role. In summary, one can choose different combinations of the steps outlined in Table 4.1. For example one can choose to carry out steps 1 and 2, steps 1, 2 and 3, steps 1, 2 and 4, or steps 1, 2, 3 and 4, based on the analysis they would like to perform.

4.3.2 Stage 2: Time Based Decomposition

Time Based Decomposition further simplifies the decomposed analysis problems in the first stage. Since the time dimension is discrete, we decompose each of the four security analysis problems above into multiple subproblems, so that each instance can be treated similar to an RBAC model. We employ two different alternative decomposition strategies – the *rule schedule strategy* and the *the role schedule strategy*. These strategies, although can analyze the same problem, provide answers to different security questions. Rule schedule strategy provides analysis for short term reachability, whereas role schedule strategy provides analysis for long term reachability. Each of the four steps of Stage 1 can be analyzed by these strategies under the state configuration and administrative rule settings depicted in Table 4.1. The time based decomposition strategies provide flexibility so that different RBAC analysis procedures can be employed as given in Table 4.2.

[2]Details given in Section 4.3.2

Table 4.2. Time Based Decomposition and Available Analyzers

Analysis Performed	Rule Schedule	Role Schedule
User Assignment	SA	Any RBAC Analyzer
Role Enabling	SA	Any RBAC Analyzer
Role Hierarchy	MSA [2]	MSA
Permission Assignment	SA	Any RBAC Analyzer

Before we provide details of these two strategies, we give the steps for each stage to be performed for some of the example security questions that we discuss in Section 4.3 in Table 4.3.

Table 4.3. The steps of analysis to be performed for different security questions given in Section 4.3

Security Question	Stage 1	Stage 2
1-a	1,2	Rule Schedule
1-c	2	Role Schedule
1-d	1,2,3	Rule Schedule
1-f	4	Role Schedule
2-c	1,2	Role Schedule
3-b	1,2,3	Rule Schedule

Rule Schedule Strategy

Rule Schedule Strategy is a state space exploration approach utilizing rule schedules (s_{rule}) to decompose the analysis into smaller problems and analyze them serially with respect to time. In this strategy, we use the RBAC analysis approach by Stoller et al. [47] extensively to explore potential states reachable in different time instances.

Let $m \in \mathcal{M}_c \subseteq \mathcal{M}$ be a subset of the rules in the analysis problem. A *constant region* $\mathcal{C}(a, b, \mathcal{M}_c)$ is a bounded time interval between $t = a$ and $t = b,\ a \leq b$ such that $\forall m \in \mathcal{M}_c,\ (a, b) \subseteq s^m_{rule}$ and $\nexists m' \notin \mathcal{M}_c$ such that $s^{m'}_{rule} \subseteq (a, b)$.

Informally, if a rule m is included in a constant region $\mathcal{C}$ then it should be valid in all time slots $\alpha \in (a, b)$, and there should not be any other rule m' that is valid in some but not all of the time slots of (a, b). In the rule schedule approach, we split the timeline from 0 to T_{MAX} into non overlapping constant regions $\mathcal{C}_i$ w.r.t the s_{rule} of the roles.

In the analysis, we trace *constant regions* $\mathcal{C}_1, \mathcal{C}_2, ...$ serially with respect to time. These regions can be seen as separate RBAC systems. However, $\mathcal{C}_{i+1}$ depends on $\mathcal{C}_i, \forall i$, which implies the output of an RBAC reachability analysis at $\mathcal{C}_i$ is an input (or initial configuration) to $\mathcal{C}_{i+1}$. Since an RBAC analysis could result in multiple configurations, then, in each *constant region*, a separate RBAC analysis should be performed for each configuration generated by the analysis done in the previous *constant region*.

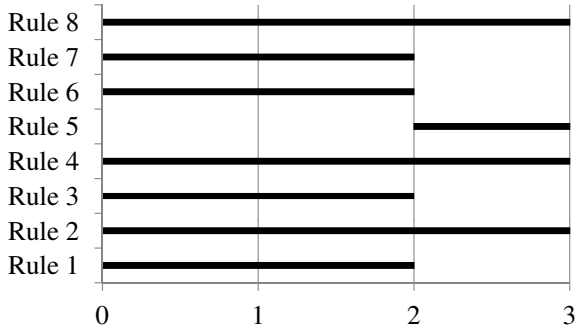


Figure 4.2. Rule Schedules

Example 4. Now, let us consider the hospital example given in Section 4.2. There are eight different administrative rules with different valid periods as depicted in Figure 4.2, where the bars indicate their respective rule schedules. As can be seen from the figure, the set of valid rules does not change in interval (0,2) $\mathcal{C}_1$ and (2,3) ($\mathcal{C}_2$). More specifically, the valid rules for $\mathcal{C}_1$ are 1, 2, 3, 4, 6, 7, 8 and the valid

rules for $\mathcal{C}_2$ are 2, 4, 5, 8. Essentially, we decompose the analysis problem of TRBAC into two subproblems which are similar to RBAC problems pertaining to these *constant regions*.

There are other issues related to role schedules that are assigned by the rules. Recall that all of the rules have a role schedule which denotes the time intervals that the role can be assigned. But, according to the rule definitions, the administrators are free to choose a sub schedule of the role schedule and assign / revoke, enable / disable and modify the role (hierarchy) schedules only for some of the designated time intervals. This further complicates the reachability analysis, since in a serial fashion, one should keep all of the possible schedule combinations for the subsequent time intervals. Therefore we make the following assumption to simplify the analysis:

Sub-schedule Assumption: For each rule, the role (or hierarchy) schedule modification operations are performed using the entire schedule s_{role} ($s_{hierarchy}$, resp.). This means that an administrator may use a rule to assign the associated role r to a user u all of the subsets of the schedule s_{role} (as long as the preconditions are satisfied). In our analysis, we assume that s_{role} ($s_{hierarchy}$, resp.) is assigned or revoked completely - no sub schedule assignments are allowed. Hence, this assumption ensures that a rule can only generate at most one (new) configuration.

Here we provide a sketch of the algorithm. The TRBAC reachability analysis starts with an initial configuration c_0 and $constant\ region\ \mathcal{C}_1$. The state space is expanded using Stoller et al.'s algorithm [47] (we refer this algorithm as SA) and

the rules that are valid at time $t = 0$ [3]. At the end of this step, a set of reachable configurations, $\mathcal{S}_1 = \{c_1, c_2, ..., c_m\}$ are obtained. Afterwards, the analysis moves to $\mathcal{C}_2$. For each distinct configuration obtained so far, SA is used to expand these configurations using the valid rules in this constant region. At the end of this step, we obtain an updated set of reachable configurations $\mathcal{S}_2 \supseteq \mathcal{S}_1$. The algorithm then moves to $\mathcal{C}_3$ and the trace goes in this fashion for a specified number of cycles P of length T_{MAX} (The algorithm returns to $\mathcal{C}_1$ whenever T_{MAX} is reached). Since TURA tuple S_T is finite and since the iterations are bounded by the number of cycles, the algorithm is guaranteed to terminate. However since this approach is a greedy heuristic, we are not guaranteed to get an optimal solution.

Role Schedule Strategy

In this approach, we split the TRBAC security analysis problem into smaller RBAC security analysis subproblems using the role schedules of the rules. The main idea is to generate subproblems $\mathcal{T}(\alpha, \mathcal{M}_s)$ for each time slot $\alpha \in (0, T_{MAX})$ with nontemporal administrative rules, so that the system can be treated like an RBAC.

Example 5. Consider Figure 4.3, which shows the role schedules of the rules in the hospital example given in Section 4.2. Here, we have three distinct time slots (Time Slot 1: (0,1), Time Slot 2: (1,2), Time Slot 3: (2,3)) with different rules. The rules for Time Slot 1 are Rule 1, 3, 4, 6, and 7; for Time Slot 2 are Rule 3, 4 and 6; for Time Slot 3 are Rule 2, 4, 5, 6, and 8.

[3] For the analysis of Dynamic Temporal Role Hierarchies, certain modifications are required as given in Section 4.3.2

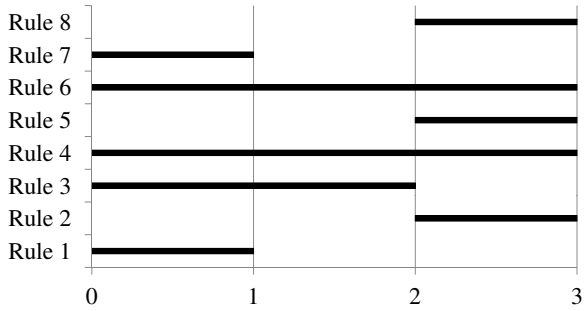


Figure 4.3. Role Schedules

In order to achieve nontemporal administrative rules, (and hence an RBAC system for each time slot), we need to remove two components: Rule Schedules and Role Schedules (Hierarchy Schedules) and we need to show the inter-time slot independency. The removal of the role schedules follows the definition of subproblems $\mathcal{T}(\alpha, \mathcal{M}_s)$. For the rule schedules, we observe the Long Run Behavior property of the administrative model that we propose.

Long Run Behavior: In the long run, rule schedules of the rules can be neglected, if the system is periodic.

Here we give the intuition of this result. Rule schedules restrict the times that a particular rule can be fired. This means that if a rule m is valid in at least one time slot and if the assignment/revocation (or enabling/disabling) operation that is going to be performed m is necessary for the other rules m', one can wait until m becomes valid, and perform the necessary operation. The other rules m' can be fired next time when the system periodically repeats itself. For example, suppose that we have two roles, r_1 and r_2 and two `t_can_assign` rules $(..., (4, 10), \{\}, r_1, ...)$ and $(..., (1, 3), \{r_1\}, r_2, ...)$. The first rule states that we can use it only within $(4, 10)$; the second rule states that we can only use it within $(1, 3)$. Notice that

if the rules are serially applied with respect to time, then since the second rule has a precondition of r_1, we cannot fire second rule if we do not have r_1 already assigned. It means that first we need to wait until first rule becomes valid (until $t = 4$) and assign r_1. Then we should wait until the system restarts from $t = 0$ (since it is periodic) to fire second rule. Then the Long Run Behavior property ensures that for the reachability analysis purposes, if one waits sufficient amount of time then the effects of these kind of rule conflicts can safely be neglected. This property allows us to treat all of the rules valid on the entire time line. Hence, the s_{rule} restrictions can be relaxed from the rules.

In order to handle the independency issues among different time slots, we need to consider preconditions. Recall that we define the preconditions as (Pos, Neg) relations to be satisfied in order to execute a rule. Now consider a rule $m \in \mathcal{M}$ which belongs to $\mathcal{T}(\alpha, \mathcal{M})$, and $\hat{s} = \alpha$. In order to execute m, the precondition relations declared by (Pos, Neg) of m must be satisfied for $\hat{s}$. For each role $pos \in Pos$ ($neg \in Neg$, resp.) $\hat{s} \subseteq s_{pos}$ ($\hat{s} \cap s_{pos} = \emptyset$, resp.) must be satisfied, which simply depends on the corresponding (single) time slot in s_{pos} (s_{neg}, resp.). Then it is sufficient to check the schedule only for time slot α for each rule. This implies that the preconditions do not depend on other time slots, hence the time slots are independent.

So, using the Long Run Behavior property and the independency of time slots, one can perform an RBAC reachability analysis using the rules $m \in \mathcal{M}$ for time slot α. Then, the whole TRBAC system can be analyzed by a series of independent RBAC systems $\mathcal{T}_i$ traced separately. This reduction provides usability of any RBAC reachability analysis procedure proposed in the literature.

The computational complexity of the algorithm depends on the RBAC analyzer. Suppose that the RBAC analyzer has the complexity $O(\cdot)$ then our approach yields a complexity of $O(T_{MAX}\cdot)$ since we utilize the RBAC analyzer for each time slot (Totally we have T_{MAX} of them). Since the algorithm runs for T_{MAX} iterations and given that the RBAC analyzer terminates, our algorithm is guaranteed to terminate.

Modified SA for Hierarchy Analysis

The RBAC Analysis algorithm proposed by Stoller et al. [47] is a state space exploration algorithm which is proved to be fixed parameter tractable. In our decomposition approach, the subproblems obtained by the decomposition can be analyzed by SA for Role Enabling, User to Role and Permission to Role assignment relations. However, due to the precondition structure and SA not capable of handling the `can_modify` rule, SA is unable to analyze the Temporal Role Hierarchy subproblem. In this section, we make certain modifications on SA to fit the requirements of the role hierarchy analysis strategy that we propose for a `TURA` analysis instance. We call this modified algorithm as MSA, which is still a state space exploration algorithm, specifically designed for role hierarchies. The purpose of MSA is to generate different possible static role hierarchies given a set of `t_can_modify` rules. This algorithm can be used in both Rule Schedule and Role Schedule strategies.

The state space is composed of the TRBAC configurations c, represented by $DTRH$, generated by moves m, and authorized by the rules $\mathcal{M}$. In the configurations, the precondition statements are crucial to determine the relationship among

different rules. A role is *hierarchy negative*, if it appears negated in either junior or senior preconditions of a `t_can_modify` rule. The other roles are called *hierarchy non-negative*. A role is *hierarchy positive*, if it appears positive in either junior or senior preconditions of a `t_can_modify` rule. The other roles are called *hierarchy non-positive*. Any move m related to a DTRH policy with hierarchy non-negative or hierarchy non-positive roles is called an invisible transition, the others are called visible transition. Any invisible transition that creates a conflict with the anti-symmetric property of DTRH in Section 4.2 generates a new state. Any visible transition that creates a conflict with the anti-symmetric property of DTRH in Section 4.2 is prohibited.

In the analysis for role hierarchies, there is no goal state to be achieved, rather all possible hierarchy configurations are constructed to be used to interpret the implicit role assignments of the other steps of the analysis.

4.3.3 Stage 3: Interpretation of the Results

The final step of the security analysis is to interpret the results obtained to conclude whether the access control configuration is vulnerable based on the analysis of interest. In our analysis methodology, each step of the four step analysis procedure outputs results for a different relation in TRBAC. However, these results are not sufficient individually to answer the security questions. The results of different steps of the analysis should be utilized together to obtain the correct result. For instance, Role Assignment analysis could state that the goal role would be assigned to the target user, but that role might not get enabled at that time instance, meaning that it is not possible for that particular user to exercise the goal role.

Algorithm 1 The Modified Stoller et al.'s Algorithm (MSA)

Input: Rules $m \in \mathcal{M}_{ALL}$ and initial state c_0
Output: $\mathcal{S}_\mathcal{P}$
Set $\mathcal{S}_\mathcal{T} = \{c_0\}$ as temporary ,$\mathcal{S}_\mathcal{P} = \emptyset$ as permanent set
Determine the non-positive and non-negative roles
while $\mathcal{S}_\mathcal{T} \neq \emptyset$ **do**
  Get a state $c \in \mathcal{S}_\mathcal{T}$
  Create a temporary state $c_{temp} = c$
  for all Rules $m \in \mathcal{S}_\mathcal{T}$ that generate an invisible transition **do**
    Check for hierarchy conflicts in c_{temp}
    if There exists any violation **then**
      Create a new state c'
      Apply rule m on c'
      Set $\mathcal{S}_\mathcal{T} = \mathcal{S}_\mathcal{T} \cup c'$
    else
      Apply rule m on c_{temp}
    end if
  end for
  Set $c = c_{temp}$
  for all Rules $m \in \mathcal{S}_\mathcal{T}$ that generate a visible transition **do**
    Create a new state c'
    Check for hierarchy conflicts in c'
    if There exists any violation **then**
      Discard c'
    else
      Set $\mathcal{S}_\mathcal{T} = \mathcal{S}_\mathcal{T} \cup c'$
    end if
  end for
  Set $\mathcal{S}_\mathcal{T} = \mathcal{S}_\mathcal{T} \backslash c$
  Set $\mathcal{S}_\mathcal{P} = \mathcal{S}_\mathcal{P} \cup c$
end while

This step is crucial to interpret the security properties correctly.

Suppose that all four steps of the analysis is done. Each step outputs a set of state configurations denoted as C_1, C_2, C_3 and C_4 respectively for the four steps. Each configuration $c \in C_1$ is composed of TUA, $c \in C_2$ is composed of RS, $c \in C_3$ is composed of $DTRH$ and $c \in C_4$ is composed of $DTRH$ policies. For notational simplicity, we denote the relations as configurations. Under these settings a given TRBAC policies and rules create a security violation if they satisfy the following criteria for different security questions of interest:

- Explicit Role Assignment: $\exists TUA \in C_1, RS \in C_2 : (u, r, s_1) \in TUA \wedge (r, s_2) \in RS \wedge s_1 \cap s_2 \neq \emptyset$.
- Implicit Role Assignment: $\exists TUA \in C_1, RS \in C_2, DTRH \in C_3 : (u, r_1, s) \in TUA \wedge (r_1 \langle \mathcal{F} \rangle_{s_{i_1}} r_2), ..., (r_n \langle \mathcal{F} \rangle_{s_{i_n}} r) \in DTRH \wedge (r_1, s_{j_1}), (r_2, s_{j_2}), ..., (r, s_{j_n}) \in RS \wedge s \cap s_{i_1} \cap ... \cap s_{i_n} \cap s_{j_1} \cap ... \cap s_{j_n} \neq \emptyset$ [4]
- Role Enabling $\exists RS \in C_2 : (r, s) \in RS \wedge s \neq \emptyset$
- Permission Assignment $\exists TPA \in C_4 : (p, r, s) \in TPA \wedge s \neq \emptyset$
- Liveness for Explicit Role Assignment: $\forall s_1, s_2, \nexists TUA \in C_1, RS \in C_2 : (u, r, s_1) \notin TUA \vee (r, s_2) \notin RS$.
- Mutual Exclusion for Explicit Role Assignment: $\exists TUA \in C_1, RS \in C_2 : (u_1, r, s_1) \in TUA \wedge (u_2, r, s_2) \in TUA \wedge (r, s_3) \in RS \wedge s_1 \cap s_2 \cap s_3 \neq \emptyset$

[4] Depending on the type of role hierarchy, role enabling criteria must satisfy the DTRH properties given in Definitions 6,7,8,9.

4.4 TRBAC Computational Experiments

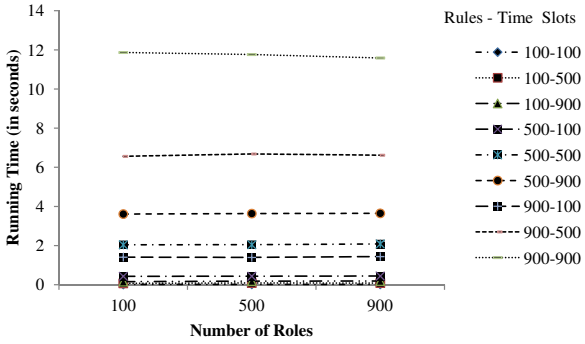
We have performed computational experiments for the analysis of TRBAC using Rule and Role Schedule Approaches. In our experiments we demonstrated the performance of the Role Assignment (Step 1) and Role Hierarchy (Step 3), since the other steps are analogous to Step 1. In the experiments we employ SA and MSA for Role Assignment and Role Hierarchy components.

We implement our algorithm with C programming language and run it on a computer with 3 GB RAM and Intel Core2Duo 3.0 GHz processor running Debian Linux operating system. In the experiments, the initial state is set to be an empty state (meaning that there are no role assignments), and the rules and the goal are created randomly by the code with respect to the corresponding parameter values for the number of rules, number of roles, number of time slots and the number of cycles. As we discussed before, we assume separate administration. Also, for role hierarchies, we assume a general hierarchy relation. The parameter settings are shown on Table 4.4. 10 replications are done for each parameter setting and their average is reported. The results are in Figure 4.4(a),4.4(b) and 4.4(c).

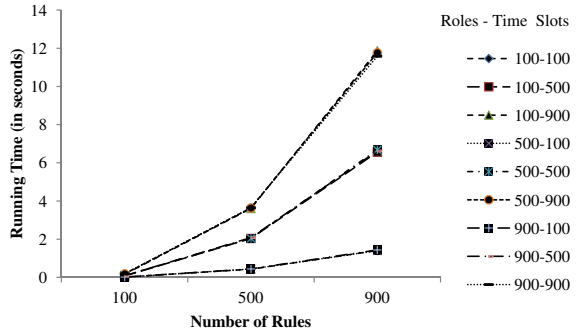
Table 4.4. Parameter Settings

Number of Roles $\|R\|$	100, 500, 900
Number of Rules $\|\mathcal{M}_{ALL}\|$	100, 500, 900
Number of Time Slots T_{MAX}	100, 500, 900
Number of Cycles P	30 for all cases

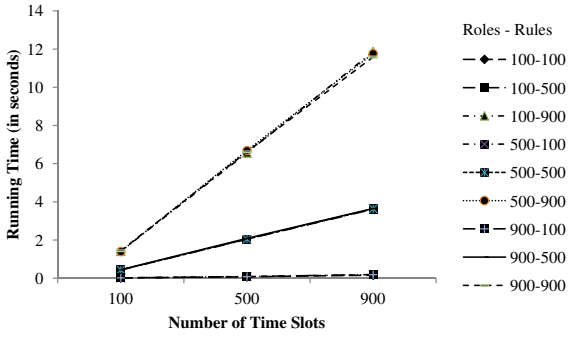
Although in the experiments we demonstrate the time complexity of our approaches with respect to changes in the system parameters, we were unable to obtain real-life data sets. As outlined by Schaad et al. [45], in a real-life large



(a) Effect of Number of Roles

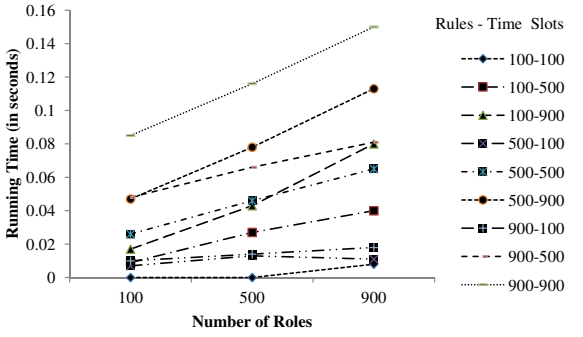


(b) Effect of Number of Rules

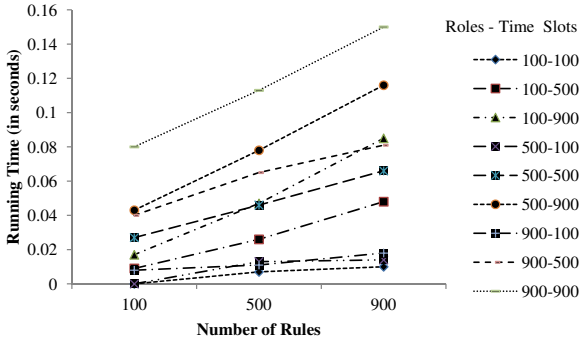


(c) Effect of Number of Time Slots

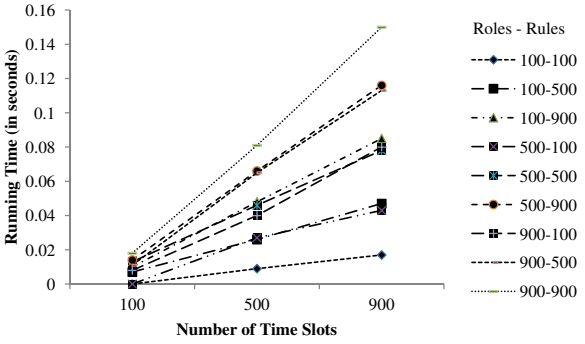
Figure 4.4. Rule Schedule Approach for Role Assignment



(a) Effect of Number of Roles



(b) Effect of Number of Rules



(c) Effect of Number of Time Slots

Figure 4.5. Role Schedule Approach for Role Assignment

enterprise, the average ratio of roles to users is about 3.2% with an average number of roles being around 1300. Also as presented by Ferrara et al. [21], ARBAC policies for the Hospital dataset has 13 roles and 37 rules, University has 32 roles and 449 rules and a typical Bank has around 500 to 1300 roles and 2000 to 8000 rules. As demonstrated in our experiments below, our approaches are scalable to cover systems of this magnitude.

4.4.1 User to Role Assignment Experiments

The complexity of the **rule schedule approach** algorithm depends not only on the number of roles and rules but also depends on the number of time slots, and the schedules (rule-role) that are assigned to the roles. The state space that is generated by this algorithm tends to be exponential in the worst case since it is a brute force state space exploration algorithm.

According to the results obtained for the rule schedule approach, the run time performances of the algorithms do not tend to be exponential, especially for the number of roles. A possible explanation to this situation is that the datasets are generated randomly. Hence there does not exist any "pattern" among the rules. We mean pattern in the sense that, the components that determine the usability of the rules, i.e., all of the precondition relations, rule and role schedules of the moves are generated randomly – so it might become probabilistically harder to satisfy all of these conditions. Nevertheless, the results give some insight about how the algorithm is likely to behave under different parameter settings.

The effect of number of rules while all other parameters are constant is more significant and tends to be an increasing relationship as number of rules increases

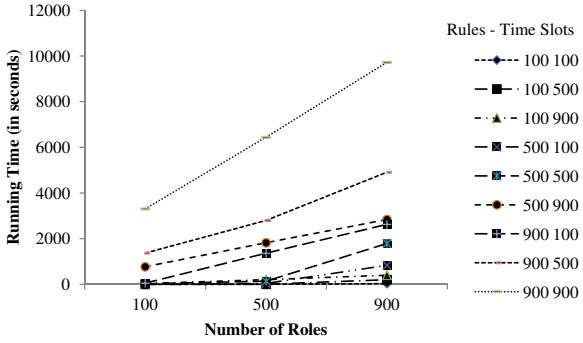
(See Figure 4.4(b)). Moreover, the increasing tendency becomes more significant as the number of roles and number of time slots increase. Furthermore, there is a noticeable group formation between the fixed parameters (number of roles and number of time slots). The groups are formed by different number of time slots values indicating that the effect of number of roles is comparably smaller. Finally, Figure 4.4(c) denotes the relationship between different values of number of time slots parameter when the other two parameters are kept constant. The results show that for the majority of the cases, there is a linearly increasing relationship with the increasing number of rules.

For the **role schedule approach**, we use SA. According to the results obtained, there is a linear and increasing relationship with 100, 500 and 900 roles in the system while all other parameters are constant (See Figure 4.5(a)). The effect of number of rules while all other parameters are constant is very similar to the effect of roles. There is an increasing relationship in the running time as the number of rules increases (See Figure 4.5(b)).

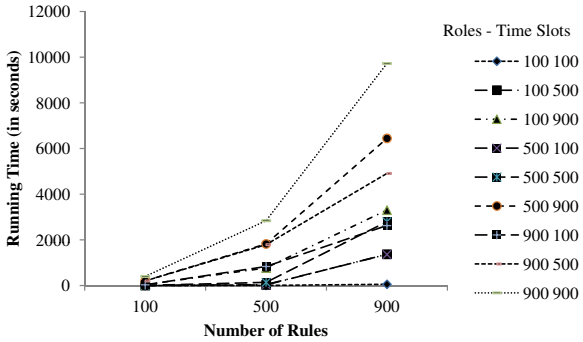
Finally, Figure 4.5(c) denotes the relationship between different values of number of time slots parameter when the other two parameters are kept constant. The results show that there is a linearly increasing behavior as the number of time slots increase. This result is expected since the complexity of the algorithm linearly depends on this parameter.

4.4.2 Role Hierarchy Experiments

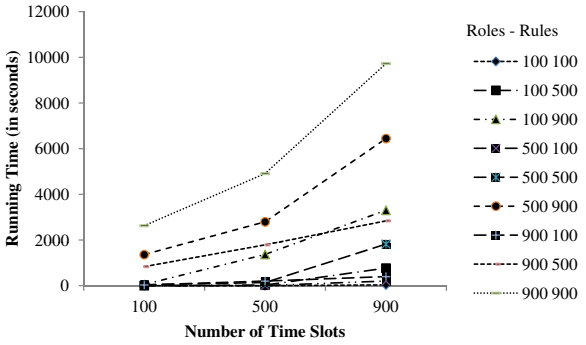
In the role hierarchy experiments, we observe that the running times of both of the approaches increased significantly. Especially for higher parameter settings



(a) Effect of Number of Roles



(b) Effect of Number of Rules



(c) Effect of Number of Time Slots

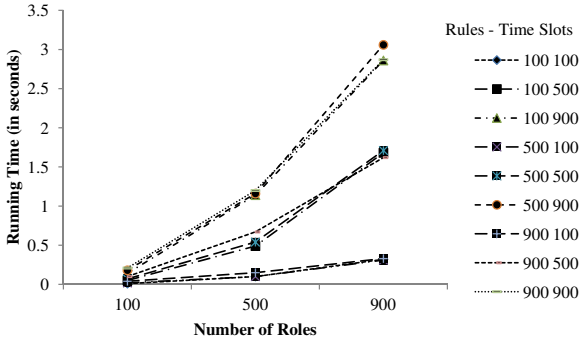
Figure 4.6. Rule Schedule Approach for Role Hierarchy

for Rule Schedule Approach, running times of 10000 seconds, as opposed to a maximum of 12 seconds for User to Role Assignment experiments are observed. The underlying reasoning for this drastic increase is the fact that the state space consists of a pair of roles. Moreover, the process of determining whether an intended update in any of the role hierarchy pairs require examining the existing role hierarchy pairs to make sure that the newly imposed changes will not create a conflict.

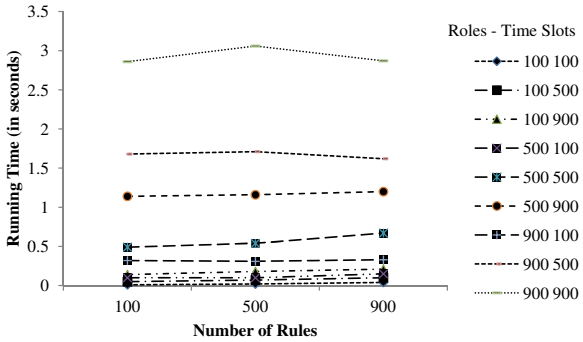
When the run time performances of rule schedule and role schedule approaches are compared, a similar pattern as in the User to Role Assignment experiments is observed. Role Schedule approach is significantly faster than the Rule Schedule approach due to the fact that the Rule Schedule approach is an exponential state space exploration algorithm. The experimental results are given in Figures 4.6(a),4.6(b) and 4.6(c) for Rule Schedule and Figures 4.7(a),4.7(b) and 4.7(c) for Role Schedule approach.

4.5 Spatio-temporal RBAC Extension

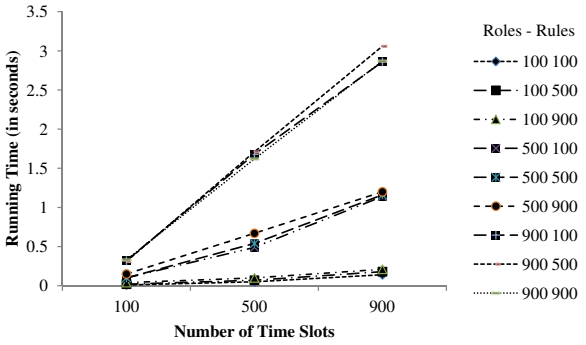
Spatio-temporal RBAC model imposes an additional dimension to the TRBAC, which is the location information, and the access decisions are made considering not only the time and assignments but also the location from which the access request is submitted. This adds another layer of complexity on the TRBAC model, as though location layer could also be simplified in a similar manner that is used for the temporal dimension. In this section, we extend our proposed model and security analysis for TRBAC to handle rights leakage problem in STRBAC. As a result of the security analysis in STRBAC, the following potential security ques-



(a) Effect of Number of Roles



(b) Effect of Number of Rules



(c) Effect of Number of Time Slots

Figure 4.7. Role Schedule Approach for Role Hierarchy

tions could be addressed:

1. **Safety:**
   (a) Will there be no reachable state in which a user u is assigned to a role r at location l?
   (b) Will an enabled role r eventually be disabled at location l?
   (c) Will a user u ever assigned to a role r at any location?
2. **Liveness:**
   (a) Will an enabled role remain enabled at location l?
   (b) Will a user u eventually be assigned to a role r at location l?
3. **Mutual Exclusion:** Will a user u be assigned to roles r_1 and r_2 at the same location?

In our extended model, we embed spatial data to user-role assignment, permission-role assignment and role status relations. Next, we make necessary definitions towards this end, and propose our extended model.

4.5.1 Spatio-temporal RBAC and its Administrative Model

In modeling STRBAC, the primary challenge is to determine how the location dimension is embedded. There are two distinct ways that the locations can be represented. The locations could either be *physical*, which means the longitudes and latitudes based on a GPS location, or they could be *logical* locations semantics of the location is used, like the name of the building, department, the names of

the cross streets of an intersection. Logical locations can be considered as labels attached to certain places. Depending on the granularity of the access control decisions, either one could be implemented. However we choose to implement the logical locations because of the following reasons. First, the physical location are very fine grained so that this granularity would be considered too excessive for many practical purposes. For example, the approximately 1984 physical locations coincide with the location of the Rutgers Business School at 1 Washington Park, Newark, NJ, 07102, USA[5]. Second, as the security analysis of RBAC with only the temporal dimension is not scalable itself (See Rule Schedule approach), it is intuitive that using a very large set of physical locations will also be not scalable. Therefore, we utilize logical locations in our analysis. Before we define the logical locations, let us first define the physical locations as tuples of longitudes and latitudes $\langle lon, lat \rangle$ and let $P = \{\langle lon, lat \rangle\}$ is the set of physical locations.

Definition 12. (Logical Locations) Let L be a bounded set of labels $\hat{l_1}, \hat{l_2}, \hat{l_3}, ..., \hat{l_n}$. The mapping function $f : P \rightarrow L$ is defined as a many-to-one labeling of physical locations into logical locations. Any subset $l \subseteq L$ is defined as a collection of locations.

The definition of the mapping function f, i.e. mapping *which* physical locations to *which* logical locations, depends solely on the locations that a particular STRBAC is authorized to control. Therefore it varies among different companies and access control systems. But the structure of the function being a many-to-one is common for all applications.

[5]Reference: Google Maps at http://maps.google.com/

The mapping function will reduce the amount of locations considerably since a subset of physical locations will be mapped into a single logical location. Without loss of generality, this mapping also satisfies the situation of a logical location for a physical location.

The logical locations are embedded into the model using the spatial-schedules, defined as follows:

Definition 13. (Spatial Schedules) The schedules with spatial dimension is a pair $\langle s, l\rangle \in SL$ that is belongs to the set of spatial schedules $SL \subseteq S \times L$.

The any pair $\langle s, l\rangle \in SL$ can be represented shortly as $sl \in SL$. A pair of time slot t and label $\hat{l}$, written as $\langle t, \hat{l}\rangle$ is an element of $\langle s, l\rangle$ ($\langle t, \hat{l}\rangle \in \langle s, l\rangle = sl$). Now we can make the definition for a Spatio-temporal RBAC configuration.

Definition 14. STRBAC Configuration. Let SL be the set of all possible spatial schedules over T_{MAX} and $|L|$. An STRBAC configuration over is a tuple $M = \langle U, R, PRMS, STUA, STPA, SRS\rangle$ where U, R and $PRMS$ are finite sets of *users*, *roles*, and *permissions*, respectively, $STUA \subseteq (U \times R \times SL)$ is the *spatio-temporal user to role assignment* relation, $STPA \subseteq (PRMS \times R \times SL)$ is the *spatio-temporal permission to role assignment* and the relation, $SRS \subseteq (R \times SL)$ is the *spatial-role-status* relation.

A tuple $(u, r, sl) \in STUA$ represents that user u is a member of the role r only during the time intervals and at the locations depicted by the spatial schedule sl. A role can be either enabled or disabled. A tuple $(r, sl) \in SRS$ imposes that role r is *enabled* only during the time intervals and at the locations provided by

sl, and *disabled* otherwise. A tuple $(p, r, sl) \in STPA$ means that permission p is associated to role r only in the time intervals and at the locations denoted by sl. Thus, a user u is granted permission p at time $t \in [0, T_{MAX}]$ and location $\hat{l}$ provided that there exists a role $r \in R$ such that $(u, r, sl_1) \in STUA$, $(r, sl_2) \in RS$, $(p, r, sl_3) \in SPA$, and $\langle t, \hat{l} \rangle \in (sl_1 \cap sl_2 \cap sl_3)$, for some spatial schedules sl_1, sl_2 and sl_3.

We assume that relation SRS for each role $r \in R$ contains always exactly one pair with first component r. Similarly, the relation $STUA$ contains exactly one tuple for each pair in $U \times R$. Thus, if a role r is disabled in any time interval for a location $\hat{l}$, we require that SRS relates r with the empty schedule for $\hat{l}$. Similarly, if a user u does not belong to a role r in any time interval at a given location l, the pair (u, r) is associated to the empty schedule for $\hat{l}$ by the relation $STUA$.

Now, we can present our administrative model that allows administrators to make changes to the spatial role-status relation SRS, spatio-temporal user-role assignment relation $STUA$, spatio-temporal permission-role assignment relation $STPA$ by using enable / disable, and assignment / revocation rules, respectively. The goal of these rules is to update the intervals of the spatial schedule sl associated to the corresponding relation.

Similar to the analysis of TRBAC model, in the analysis of the STRBAC model, we assume that the analysis for $STPA$ can be made separately, since it is not directly related to the analysis of other components in terms of the security questions in consideration. Therefore, we define the spatio-temporal `URA` and spatio-temporal `PRA` systems separately to observe the state transitions.

Definition 15. (Spatio Temporal User to Role Administration). A STURA system is a tuple $\mathcal{S}_{_T} = \langle M, \texttt{s\_can\_enable}, \texttt{s\_can\_disable}, \texttt{st\_can\_assign}, \texttt{st\_can\_revoke}\rangle$ where $M = \langle U, R, PRMS, STUA, STPA, SSRS\rangle$ is an STRBAC policy over SL, and $\texttt{s\_can\_enable}, \texttt{s\_can\_disable}, \texttt{st\_can\_assign}, \texttt{st\_can\_revoke} \subseteq (R \times SL \times 2^R \times 2^R \times SL \times R)$.

Definition 16. (Spatio Temporal Permission to Role Administration). A STPRA system is a tuple $\mathcal{S}_{_T} = \langle M, \texttt{st\_can\_assignp}, \texttt{st\_can\_revokep}\rangle$ where $M = \langle U, R, PRMS, STUA, STPA, SRS\rangle$ is an STRBAC policy over SL, and $\texttt{st\_can\_assignp}, \texttt{st\_can\_revokep} \subseteq (R \times SL \times 2^R \times 2^R \times SL \times R)$.

A *configuration* of $\mathcal{S}_{_T}$ for STURA is a tuple $(SRS, STUA)$, which is *initial* if $SRS = SRS_0$, and $STUA = STUA_0$. Similarly, a *configuration* of $\mathcal{S}_{_T}$ for STPRA is a singleton $(STPA)$, which is *initial* if $STPA = STPA_0$. Given two $\mathcal{S}_{_T}$ configurations $c = (SRS, STUA)$ and $c' = (SRS', STUA')$ for STURA' and $c = (STPA)$ and $c' = (STPA')$ for STPRA, we describe below the conditions under which there is a *transition* (or *move*) from c to c' at time $t \in \mathbb{N}$ with rule $m \in \mathcal{M}_{ALL} = (\texttt{s\_can\_enable} \cup \texttt{s\_can\_disable} \cup \texttt{st\_can\_assign} \cup \texttt{st\_can\_revoke} \cup \texttt{st\_can\_assignp} \cup \texttt{st\_can\_revokep})$, denoted $c \xrightarrow{(\tau_m, t)} c'$.

Before defining the transition relation, we first describe the components of move $m = (admin, sl_{rule}, Pos, Neg, sl_{role}, r)$. Move m can be executed only by a user, say ad, belonging to the *administrative role* $admin \in R$.

The times t and locations $\hat{l}$ in which ad can execute m are all those in which ad is assumed to be a member of role $admin$, and furthermore, $\langle t, \hat{l}\rangle$ must also belong to the schedule sl_{rule} which denotes the time intervals and locations when

m can be performed (or we say *valid*): $\langle t, \hat{l} \rangle \in (sl_{ad} \cap sl_{admin} \cap sl_{rule})$ where $(ad, admin, sl_{ad}) \in STUA$ and $(admin, sl_{admin}) \in SRS$. In the rest of the section we say that m can be *executed* at time t and location $\hat{l}$ whenever $\langle t, \hat{l} \rangle$ fulfills the above condition. The component sl_{role} is used to update the schedule of a role, or the membership of a user to a role, depending on the kind of rule of m. The pair of disjoint role sets (Pos, Neg) is called the *precondition* of m whose fulfillment depends by the kind of the rule m.

The fulfillment of the precondition of a spatial can-enable and spatial can-disable rule depends on the current status of the other roles. Let $\hat{sl} \subseteq sl_{role}$. A spatial can-enable or spatial can-disable rule $m = (admin, sl_{rule}, Pos, Neg, sl_{role}, r)$ satisfies its precondition (Pos, Neg) w.r.t. candidate spatial schedule $\hat{sl}$, if for every time-location pair $\alpha \in \hat{sl}$, if (1) for every role $pos \in Pos$, $\alpha \subseteq sl_{pos}$ where $(pos, sl_{pos}) \in RS$, (2) for every role $neg \in Neg$, $\alpha \cap sl_{neg} = \emptyset$, where $(neg, sl_{neg}) \in RS$, and (3) α satisfies all preconditions. In other words, a candidate schedule $\hat{sl} \subseteq sl_{role}$ satisfies a precondition only if each time-location pair $\alpha \in \hat{s}$ satisfies the precondition individually. Let $(r, sl) \in SRS$.

Enabling Rules: A spatial can-enable rule adds a new spatial schedule to a specific role. A tuple $(admin, sl_{rule}, Pos, Neg, sl_{role}, r) \in$ `s_can_enable` allows to update the tuple $(r, sl) \in SRS$ to $(r, sl \cup \hat{sl})$ for some spatial schedule $\hat{sl}$, provided that m can be executed at time t, at location $\hat{l}$ and also satisfies its precondition. Formally, rule m is executable at time t and location $\hat{l}$, m satisfies its precondition (Pos, Neg) w.r.t.spatial schedule $\hat{sl}$, $SRS' = (SRS \setminus \{(r, sl)\}) \cup \{(r, sl \cup \hat{sl})\}$, $STPA' = STPA$ and $STUA' = STUA$.

Disabling Rules: A spatial can-disable rule removes a spatial schedule from a

designed role. A tuple $m = (admin, sl_{rule}, Pos, Neg, sl_{role}, r) \in$ `s_can_disable` allows to update the tuple $(r, sl) \in SRS$ to $(r, sl \setminus \hat{sl})$, for some spatial schedule $\hat{sl}$, provided that m can be executed at time t, at location $\hat{l}$ and satisfies its precondition. Formally, m is executable at time t and location $\hat{l}$, m satisfies its precondition (Pos, Neg) w.r.t. spatial schedule $\hat{sl}$, $SRS' = (SRS \setminus \{(r, sl)\}) \cup \{(r, sl \setminus \hat{sl})\}$, $STPA' = STPA$ and $STUA' = STUA$.

The next two rules are similar to those given above with the difference that we now update the spatial schedules associated to each element of the spatial user-role assignment relation. Another difference is that spatio-temporal can-assign and spatio-temporal can-revoke rules have a different semantics to fulfill their preconditions. A user $u \in U$ satisfies a precondition (Pos, Neg) w.r.t. a spatial schedule $\hat{st}$ if for every time-location pair $\alpha \in \hat{sl}$, (1) for every $(u, pos, sl_{pos}) \in STUR$ with $pos \in Pos$, $\alpha \subseteq sl_{pos}$, (2) for every $(u, neg, sl_{neg}) \in STUA$ with $neg \in Neg$, $\alpha \cap sl_{neg} = \emptyset$, and (3) α satisfies all preconditions. Let $(u, r, sl) \in STUA$.

Assignment Rules: A tuple $(admin, sl_{rule}, Pos, Neg, sl_{role}, r) \in$ `st_can_assign` allows to update the spatio temporal user-role assignment relation for the pair (u, r) as follows. Let $\hat{sl}$ be a spatial-schedule over SL with $\hat{sl} \subseteq sl_{role}$. Then, if m can be executed at time t, and location $\hat{l}$, and user u satisfies the precondition (Pos, Neg) w.r.t. spatial schedule $\hat{sl}$, then the tuple (u, r, sl) is updated to $(u, r, sl \cup \hat{sl})$, i.e. $STUA' = (STUA \setminus \{(u, r, sl)\}) \cup \{(u, r, sl \cup \hat{sl})\}$, $STPA' = STPA$ and $SRS' = SRS$.

Revocation Rules: A tuple $(admin, sl_{rule}, Pos, Neg, sl_{role}, r) \in$ `st_can_revoke` allows to update the spatio-temporal user-role assignment relation for the pair

(u, r) as follows. Let $\hat{sl}$ be a schedule over SL with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and location $\hat{l}$ and user u satisfies the precondition (Pos, Neg) w.r.t. spatial schedule $\hat{sl}$, then the tuple (u, r, sl) is updated to $(u, r, sl \setminus \hat{sl})$, i.e. $STUA' = (STUA \setminus \{(u, r, sl)\}) \cup \{(u, r, sl \setminus \hat{sl})\}$, $STPA' = STPA$ and $SRS' = SRS$.

The rules for updating the spatio-temporal permission-role assignment is again similar to the user-role assignment rules. A permission $p \in PRMS$ satisfies a precondition (Pos, Neg) w.r.t. a spatial schedule $\hat{sl}$ if for every time-location pair $\alpha \in \hat{sl}$, (1) for every $(p, pos, sl_{pos}) \in STPA$ with $pos \in Pos$, $\alpha \subseteq sl_{pos}$, (2) for every $(p, neg, sl_{neg}) \in STPA$ with $neg \in Neg$, $\alpha \cap sl_{neg} = \emptyset$, and (3) α satisfies all preconditions. Let $(p, r, sl) \in STPA$.

Assignment Rules: A tuple $(admin, sl_{rule}, Pos, Neg, sl_{role}, r) \in$ `st_can_assignp` allows to update the spatio-temporal permission-role assignment relation for the pair (p, r) as follows. Let $\hat{sl}$ be a spatial schedule with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and location $\hat{l}$ and permission p satisfies the precondition (Pos, Neg) w.r.t. spatial schedule $\hat{sl}$, then the tuple (p, r, sl) is updated to $(p, r, sl \cup \hat{sl})$, i.e. $STPA' = (STPA \setminus \{(p, r, sl)\}) \cup \{(p, r, sl \cup \hat{sl})\}$, $STUA' = STUA$ and $SRS' = SRS$.

Revocation Rules: A tuple $(admin, sl_{rule}, Pos, Neg, sl_{role}, r) \in$ `st_can_revokep` allows to update the spatio-temporal permission-role assignment relation for the pair (p, r) as follows. Let $\hat{sl}$ be a spatial schedule over SL with $\hat{sl} \subseteq sl_{role}$. Then, if m can be executed at time t, and location $\hat{l}$ and permission p satisfies the precondition (Pos, Neg) w.r.t. spatial schedule $\hat{sl}$, then the tuple (p, r, sl) is

updated to $(p, r, sl \setminus \hat{sl})$, i.e. $STPA' = (STPA \setminus \{(p, r, sl)\}) \cup \{(p, r, sl \setminus \hat{sl})\}$, $STUA' = STUA$ and $SRS' = SRS$.

Example 6. Consider once again the hospital example given in Section 4.2. Suppose that the hospital has two different locations, called LOC-1 and LOC-2, where LOC-1 is the headquarters and LOC-2 is a branch. Assume that there are 7 different roles, namely, Employee (EMP), Day Doctor (DDR), Night Doctor (NDR), Practitioner (PRC), Nurse (NRS), Secretary (SEC) and Chairman (CHR). Hospital works for 24 hours and there are three different shifts (time slots) from 8 am to 4 pm (Time Slot 1), 4 pm to 12 am (Time Slot 2) and 12 am to 8 am (Time Slot 3). Only the Chairman role (CHR) has administrative privileges. Different administrative rules are deployed for these locations.

1. (CHR, $\{(0,2)\} - \{LOC-1\}$, $\{DDR\}$, $\emptyset$, $\{(0,1)\} - \{LOC-1, LOC-2\}$, PRC)$\in$ `can_enable`: At time slots 1 and 2, and location LOC-1, a chairman can enable the role *Practitioner* for the first time slot of locations LOC-1 and LOC-2 if the role *Day Doctor* is also enabled during this time slot and the respective locations.
2. (CHR, $\{(0,3)\} - \{LOC-1, LOC-2\}$, $\{EMP, NDR\}$, $\{(2,3)\} - \{LOC-1, LOC-2\}, \{LOC-1\}$, NRS) $\in$ `can_disable`: At time slots 1, 2 and 3, and locations LOC-1 and LOC-2 a chairman can disable the role *Nurse* for the third time slot of locations LOC-1 and LOC-2 if the roles *Employee* and *Night Doctor* are enabled at this time slot and the respective locations.
3. (CHR, $\{(0,2)\} - \{LOC-1\}$, $\{EMP\}$, $\{NRS\}$, $\{(0,2)\} - \{LOC-1\}$, DDR) $\in$ `t_can_assign`: At time slots 1 and 2, and location LOC-1 a chair-

man can assign the role *Day Doctor* for the first and the second time slots of location LOC-1 to any user that has *Employee* role and does not have *Nurse* role during these time slots and location.

4. $(CHR, \{(0,3)\} - \{LOC-1, LOC-2\}, \emptyset, \emptyset, \{(0,3)\} - \{LOC-1, LOC-2\}, SEC) \in$ `t_can_revoke`: At time slots 1, 2 and 3, and locations LOC-1 and LOC-2 a chairman can revoke the role *Secretary* for all time slots and locations of any user that has *Secretary* role assigned in these slots and respective locations.

5. $(CHR, \{(2,3)\} - \{LOC-1\}, \{EMP\}, \{NRS\}, \{(2,3)\} - \{LOC-1\}, NDR) \in$ `t_can_assignp`: At time slot 3, and location LOC-1 a chairman can assign a permission to the role *Night Doctor* for the third time slot of LOC-1 if that permission is also assigned to *Employee* not assigned to *Nurse* role during this time slot and location.

6. $(CHR, \{(0,2)\} - \{LOC-1\}, \emptyset, \emptyset, \{(0,3)\} - \{LOC-1\}, NRS) \in$ `t_can_revokep`: At time slots 1 and 2, and location LOC-1 a chairman can revoke a permission from the role *Nurse* for all time slots of location LOC-1.

7. $(CHR, \{(0,2)\} - \{LOC-1\}, \{DDR\}, \emptyset, \{(0,1)\} - \{LOC-1\}, PRC) \in$ `t_can_assign`: At time slots 1 and 2, and location LOC-1 a chairman can assign the role *Practitioner* for the first time slot at LOC-1 to any user that has *Day Doctor* role during this time slot and location.

8. $(CHR, \{(0,3)\}, \{LOC-1\}, \{NDR\}, \emptyset, \{(2,3)\}, \{LOC-1\}, PRC) \in$ `t_can_assign`: At time slots 1, 2 and 3, and location LOC-1 a chairman

can assign the role *Practitioner* for the third time slot of location LOC-1 to any user that has *Night Doctor* role during this time slot and location.

4.5.2 Spatio-temporal RBAC Security Analysis

The analysis strategies for TRBAC discussed earlier in this chapter can also be utilized for spatio-temporal RBAC in the sense that, each $sl \in SL$ can be handled as a separate RBAC security analysis problem and each separate analysis performed on these time and location pairs can be combined to obtain the complete security analysis for the system. Therefore we use the three stage decomposition strategy proposed in Section 4.5.1 for STRBAC.

In the **Relation Based Decomposition**, we split the STRBAC security analysis into three separate and independent analysis (Table 4.5). In each of these steps, the state configurations and the administrative rule sets of the analysis problems are shaped with different relations.

Table 4.5. Subproblems, Initial Configurations and the Relations Used

Analysis Performed	State Configuration Represented By	Initial Configuration of the Analysis
1. User Assignment 2. Role Enabling	STUA SRS	STUA Relation of the target user SRS Relations of all roles
✓Role Assignment Analysis is complete.		
3. Permission Assignment	STPA	STPA Relation of the target role
✓Full Analysis is complete.		

For each different analysis, the rule set is composed of the following rules:

1. User Assignment: `st_can_assign`, `st_can_revoke`
2. Role Enabling: `s_can_enable`, `s_can_disable`

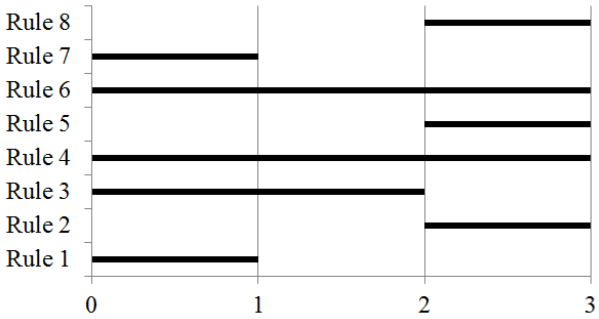
3. Permission Assignment: st_can_assignp, st_can_revokep

In the **Time Based Decomposition** the STRBAC subproblems are further simplified into RBAC security analysis problems that could be handled much easily. Although we employ two different alternative decomposition strategies – the *rule schedule strategy* and the *role schedule strategy* for the TRBAC, for STRBAC we employ only the *role schedule strategy* due to two reasons. First, since the rule schedule strategy is not scalable for TRBAC, and hence it will be not scalable for STRBAC, as well. Second, the short term reachability becomes more complex with the location dimension. In the TRBAC case, the analysis is done for a single location starting from an initial state and using the constant regions and the valid administrative rules, new states are generated. However, in the STRBAC case, handling constant regions requires a more detailed approach as traveling between different locations also take time. As the short term reachability questions focus on the reachability precisely at a given time, analysis requires computation of the actual distances between the logical locations and the time required to travel them. Hence, we leave this as our future work (Discussed in Section 6.2). Now we discuss the (spatial) role schedule strategy adapted to STRBAC analysis.

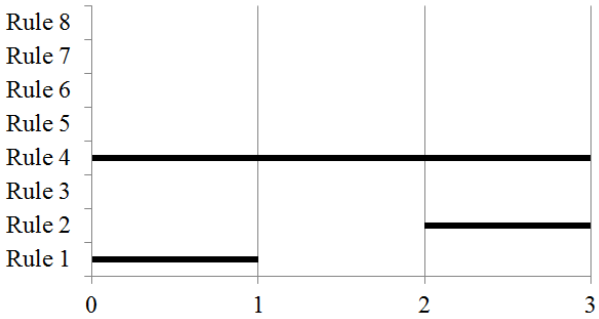
In the *role schedule strategy*, we split the STRBAC security analysis problem into smaller RBAC security analysis subproblems using the role schedules of the rules. The main idea is to generate subproblems $\mathcal{T}(\alpha, \mathcal{M}_s l)$ for each unit time-location pair $\alpha \in \langle s, \hat{l} \rangle$ with nontemporal administrative rules, so that the system can be treated like an RBAC.

Example 7. Consider Figure 4.8, which shows the role schedules of the rules in

the hospital example given in Section 4.2. Here, we have six distinct time slots and location pairs (Pair 1: (0,1)-(LOC-1), Pair 2: (1,2)-(LOC-2), Pair 3: (2,3)-(LOC-1), Pair 4: (0,1)-(LOC-2), Pair 5: (1,2)-(LOC-1), Pair 6: (2,3)-(LOC-2)) with different rules. The rules for Pair 1 are Rule 1, 3, 4, 6 and 7 ; for Pair 2 are Rule 3, 4 and 6; for Pair 3 are Rule 2, 4, 5, 6, and 8; for Pair 4 are Rule 1 and 4; for Pair 5 is Rule 4 and finally for Pair 6 are Rule 2 and 4.



(a) Role Schedule for Location LOC-1



(b) Role Schedule for Location LOC-2

Figure 4.8. Role Schedules for Example 7

The long run behavior and the independency of the time-location pairs enables us to perform an analysis using the Role Schedule approach. The long run behavior holds for spatial schedules, as well, provided that the system is periodic with respect to time and *also* the locations are visited periodically. Hence, in the long run, rule schedules of the rules can be neglected.

The time-location pairs are also independent of each other. To show this we need to consider preconditions. Recall that we define the preconditions as (Pos, Neg) relations to be satisfied in order to execute a rule. Now consider a rule $m \in \mathcal{M}$ which belongs to $\mathcal{T}(\alpha, \mathcal{M})$, and $\hat{sl} = \alpha$. In order to execute m, the precondition relations declared by (Pos, Neg) of m must be satisfied for $\hat{sl}$. For each role $pos \in Pos$ ($neg \in Neg$, resp.) $\hat{sl} \subseteq sl_{pos}$ ($\hat{sl} \cap sl_{pos} = \emptyset$, resp.) must be satisfied, which simply depends on the corresponding unit time-location pair in sl_{pos} (sl_{neg}, resp.). Then it is sufficient to check the spatial-schedule only for time-location pair α for each rule. This implies that the preconditions do not depend on other time slots, hence the time slots are independent.

So, using the Long Run Behavior property and the independency of time slots, one can perform an RBAC reachability analysis using the rules $m \in \mathcal{M}$ for time-location pair α. Then, the whole TRBAC system can be analyzed by a series of independent RBAC systems $\mathcal{T}_i$ traced separately. This reduction provides usability of any RBAC reachability analysis procedure proposed in the literature.

The computational complexity of the algorithm depends again, on the RBAC security analyzer. Suppose that the RBAC analyzer has the complexity $O(\cdot)$ then our approach yields a complexity of $O(T_{MAX}|L|\cdot)$ since we utilize the RBAC analyzer for each time-location pair (Totally we have $T_{MAX}|L|$ of them). Since the algorithm runs for $T_{MAX}|L|$ iterations and given that the RBAC analyzer terminates, our algorithm is guaranteed to terminate.

In the final stage, which is the **Interpretation of the Results**, we examine the results obtained to conclude whether the access control configuration is vulnerable to rights leakage based on the analysis of interest. In our analysis methodology,

each step of the three step analysis procedure outputs results for a different relation in STRBAC. However, these results are not sufficient individually to answer the security questions. The results of different steps of the analysis should be utilized together to obtain the correct result. For instance, Role Assignment analysis could state that the goal role would be assigned to the target user, but that role might not get enabled at that location instance, meaning that it is not possible for that particular user to exercise the goal role. This step is crucial to interpret the security properties correctly.

Suppose that all three steps of the analysis is done. Each step outputs a set of state configurations denoted as C_1, C_2, C_3 and C_4 respectively for the four steps. Each configuration $c \in C_1$ is composed of $STUA$, $c \in C_2$ is composed of SRS, $c \in C_3$ is composed of $STPA$ policies. Under these settings a given STRBAC policy and rules create a security violation if they satisfy the following criteria for different security questions of interest:

- Role Assignment: $\exists STUA \in C_1, SRS \in C_2 : (u, r, sl_1) \in STUA \wedge (r, sl_2) \in SRS \wedge sl_1 \cap sl_2 \neq \emptyset$.
- Role Enabling $\exists SRS \in C_2 : (r, sl) \in SRS \wedge sl \neq \emptyset$
- Permission Assignment $\exists STPA \in C_3 : (p, r, sl) \in STPA \wedge sl \neq \emptyset$
- Liveness for Role Assignment: $\forall sl_1, sl_2, /\exists STUA \in C_1, SRS \in C_2 : (u, r, sl_1) \notin STUA \vee (r, sl_2) \notin SRS$.

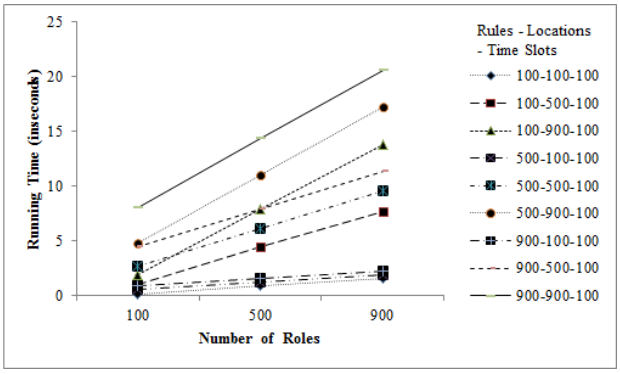
4.5.3 Spatio-temporal RBAC Computational Experiments

We tested the STRBAC extension of our algorithm with C programming language and run it on a computer with 3 GB RAM and Intel Core2Duo 3.0 GHz processor running Debian Linux operating system. In order to stay consistent with the TRBAC experiments, we set the initial state to be an empty state (meaning that there are no role assignments) for all of the locations, and the rules and the goal are created randomly by the code with respect to the corresponding parameter values for the number of rules, number of roles, number of time slots and the number of locations. We assume separate administration. The parameter settings are shown on Table 4.6. 10 replications are done for each parameter setting and their average is reported.

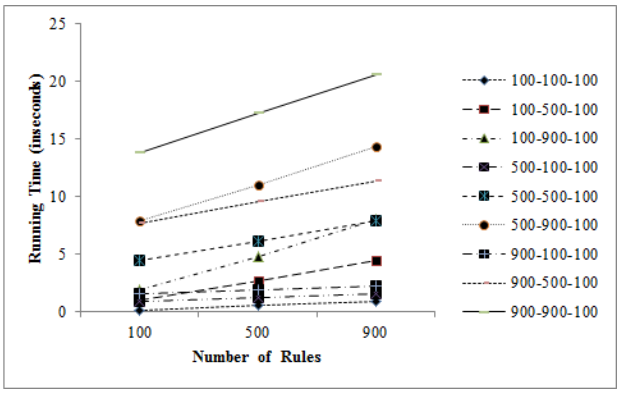
Table 4.6. Parameter Settings

Number of Roles $\|R\|$	100, 500, 900
Number of Rules $\|\mathcal{M}_{ALL}\|$	100, 500, 900
Number of Locations $\|L\|$	100, 500, 900
Number of Time Slots T_{MAX}	100 for initial experiments, 100, 500, 900 for comprehensive experiments

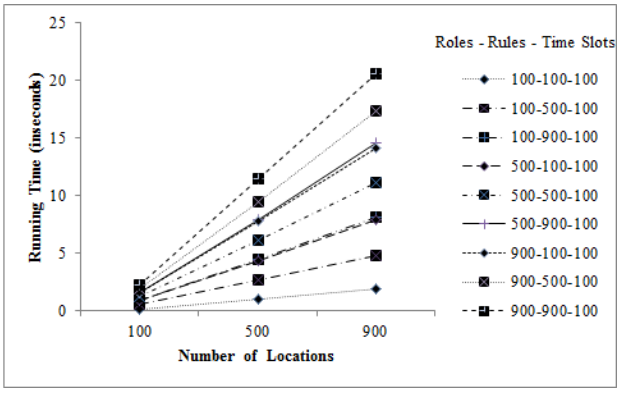
We first isolate the effect of locations in the STRBAC experiments. To accomplish this, we fix the number of time slots to be 100 for all experiments. According to the results obtained, there is a similar linear and increasing relationship with 100, 500 and 900 roles in the system while all other parameters are constant (See Figure 4.9(a)). The effect of number of rules while all other parameters are constant is very similar to the effect of roles. There is an increasing relationship in the running time as the number of rules increases (See Figure 4.9(b)). Finally, Figure 4.9(c) denotes the relationship between different number of locations when the



(a) Effect of Number of Roles



(b) Effect of Number of Rules



(c) Effect of Number of Locations

Figure 4.9. Experimental Results for STRBAC

other three parameters are kept constant. The results show that there is a linearly increasing behavior as the number of locations increase. Although these results are similar to the TRBAC experiments, the amount of time required to run the experiments is increased significantly. For example, for 900 roles, 900 rules and 100 time slots requires approximately 0.018 seconds in TRBAC experiments. In STRBAC with 900 roles, 900 rules, 100 time slots ans 100 locations this amount rises up to about 3 seconds. We can also state from the slopes of the lines in each of the three graphs, the parameter that affects the running time the most is the number of locations and the least is number of rules.

We also have comprehensive experiments, in which we show the effect of both number of time slots and the number of locations when the number of roles and number of rules kept constant. The result is given in Figure 4.10. We can state that the time required for running experiments is amplified as the number of roles and rules increase.

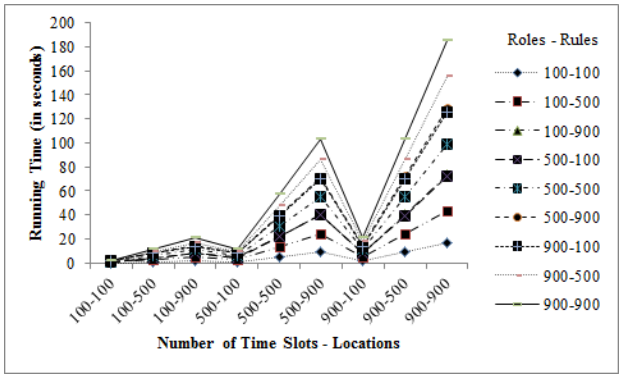


Figure 4.10. Effect of Number of Time Slots and Locations

4.6 Discussion

In this chapter we presented our methodologies to address the rights leakage problem in TRBAC and STRBAC. In summary, we have the following major contributions. We propose administrative models to control the temporal and spatio-temporal components of TRBAC and STRBAC. These models are required to perform a security analysis. Also, in the light of the earlier works related to temporal RBAC, we decide to use discretized time and location in order to simplify security analysis.

Our proposed security analysis is a very powerful decomposition based methodology that allows customization to address various different potential security questions. Also, our methodology is flexible enough to work with any RBAC security analyzer that has been proposed before. If a more efficient RBAC analyzer is proposed in the future, this RBAC analyzer could be utilized in our methodology to provide more efficient TRBAC and STRBAC analysis.

In the remainder of this section, we present some discussion about different ways achieve more efficient implementation of our security analysis methodology for TRBAC and STRBAC.

4.6.1 Temporal Role Hierarchies Execution Model

The dynamic temporal role hierarchy definition theoretically allows the access control system to have a different hierarchy at each different time slot, hence users can potentially acquire a totally different set of roles and permissions in each of these slots. Recall that, the role hierarchy set is composed of role hier-

archy policies. In fact, these policies create a tree structure with roles as nodes and the policies as the directed edges. So, the hierarchy can also be represented as a tree. In an application perspective, it is necessary to determine exactly how the temporal role hierarchies are represented in the system. There are two different ways: (1) A *separate complete hierarchy* tree for each time slot. Then, the role / permission acquisition at each time slot can be determined by tracing the complete role hierarchy tree of that particular time slot. (2) Retaining the *Hierarchy Policies* with embedded schedules, and the role / permission acquisition decisions are made on demand. Both of these approaches are useful under different circumstances. Now, we provide an insight about when to use which representation to answer a query asking whether a role is senior to another role in a given time slot. Having a separate complete role hierarchy at each time slot provides faster response to any query that checks for an implicit assignment. A simple search (like depth-first search) done on this tree will provide an efficient answer in $O(|R|log|R|)$ time. On the other hand, a search in the partial hierarchies require an exponential $O(|DTRH|^{|DTRH|})$ time. However, the partial hierarchies can be beneficial if the system faces many alterations in the role hierarchies. In this case a policy change for a single time slot requires $O(|DTRH|)$ time for the partial hierarchies, but $O(|R|log|R|)$ for separate complete hierarchy.

4.6.2 Incremental Security Analysis for TRBAC

Our proposed security analysis is a complete procedure that starts with an initial state and builds all possible state configurations based on the rules of the administrative model. Now suppose that we are interested in any change in terms of

security violations in the access control system when we add a new rule or a modification in one of the $TUA, RSTPA$ or $DTRH$ policies. More specifically, how will the output of any existing security analysis change if we perform certain modifications in the policies or rules.

We examine this problem under the user to role assignment perspective. The output of a TUA analysis (Step 1), results a set of configurations C_1 that are composed of configurations c represented by TUA. Suppose that we have the set C_1 already populated before. We have three cases to examine:

1. The initial state c_0 now has role $r*$ assigned with a schedule $s*$. Then the existing state configuration space is updated as follows:

   If there exists a `t_can_revoke` $\forall c \in C_1$, $(u, r*, s \cup s*) \in TUA$ unless the rule m that generated the state does not have $r* \in Neg$ and $s_{role}/caps* \neq \emptyset$. Then, all rules m that has $r* \notin Neg$ are executed in all states $c \in C_1$ generating new configurations based on whether the transition is visible or invisible.

2. The initial state c_0 now has role $r*$ revoked. Then the existing state configuration space is updated as follows: $\forall c \in C_1$, $(u, r*, s) \notin TUA$. Then, for any state $c \in C_1$ that is generated by a rule m that has $r* \in Pos$ is removed with any of its child states. Afterwards, all rules m are run on all remaining states in C_1 to generate new configurations.

3. The rule set now has a new rule. Then the existing state space is updated as follows: $\forall c \in C_1$, m is executed first on all of the states, and then the remaining rules are further executed on any newly generated state.

This procedure can be applied to reduce the cost of doing a full state space exploration that will generate similar states as it would do in the absence of the modifications done.

CHAPTER 5
ANALYSIS OF DATA LEAKAGE PROBLEM

Data leakage problem is similar to the rights leakage problem, since sensitive data is accessed by unauthorized users. In data leakage, the object permissions are not altered, so every object access is legal and authorized by the access control system. However, the data itself is leaked from one object to another, so that the unauthorized users would gain access to the data itself rather than directly accessing the object that has it. Therefore, examining the read and write actions of users is crucial in the process of detecting the data leakages. [1]

5.1 Overview and Challenges of the Problem

It is well known that both DAC and RBAC models suffer from their inability to prevent data leakages to unauthorized users through malware, or malicious or complacent user actions. This problem, also known as a Trojan Horse attack, may lead to an unauthorized data flow that may cause either a confidentiality or an integrity violation.

More specifically, (i) a *confidentiality violating flow* is the potential flow of sensitive information from trusted users to untrusted users that occurs via an illegal read operation, and (ii) *integrity violating flow* is the potential contamination of a

[1] The contributions in this chapter is in submission.

Table 5.1. Example Access Control Matrix

Subject	$\mathbf{o_1}$	$\mathbf{o_2}$	$\mathbf{o_3}$	$\mathbf{o_4}$	$\mathbf{o_5}$	$\mathbf{o_6}$	$\mathbf{o_7}$
$\mathbf{s_1}$	r	r	w	w	w		
$\mathbf{s_2}$	r	r	w	w	w		
$\mathbf{s_3}$			r	r	r	w	w
$\mathbf{s_4}$			r	r	r	w	w
$\mathbf{s_5}$						r	

sensitive object that occurs via an illegal write operation by an untrusted user. We now give an example to illustrate these two cases.

Example 8. Consider a DAC policy represented as an access control matrix given in Table 5.1 (r represents $read$, and w represents $write$).

Confidentiality violating flow: Suppose s_3 wants to access data in o_1. s_3 can simply accomplish this (without altering the access control rights) by exploiting s_1's read access to o_1. s_3 prepares a malicious program disguised in an application (i.e., a *Trojan Horse*) to accomplish this. When run by s_1, and hence using her credentials, the program will read contents of o_1 and write them to o_3, which s_3 can read. All this is done without the knowledge of s_1. This unauthorized data flow allows s_3 to *read* the contents of o_1, without explicitly accessing o_1.

Integrity violating flow: Suppose s_1 wants to contaminate the contents of o_6, but she does not have an explicit write access to it. She prepares a malicious program. When this is run by s_3, it will read from o_3, that s_1 has write access to and s_3 has read access, and write to o_6 using s_3's credentials, causing o_6 to be contaminated by whatever s_1 writes to o_3. This unauthorized flow allows s_1 to *write* to o_6 without explicitly accessing o_6.

Such illegal flows can occur in the many of the most common systems that

we use today because they employ DAC policies. For example, in UNIX, the key system files are only readable by root, however, the access control rights of the other files are determined solely by the users. If the root user's account is compromised, the data in the system files, such as the user account name and password hashes (/etc/passwd or /etc/shadow) could be leaked to some other users. As another example, a similar flow might occur in Social Networks as well. For instance, Facebook offers fine grained privacy settings. However they are still under the user's discretion. The third party Facebook applications that the users grant permission to access their profile could violate the user's privacy settings and access confidential information.

The first step for eliminating occurrences like the ones depicted in the example above is to perform a *security analysis*. To date, existing solutions to address such problems give the impression that such unauthorized flows could only be efficiently prevented in a dynamic setting (i.e., only by examining the actual operations), while preventing them in a static setting (i.e., by examining the authorization specifications) would require the computation of the transitive closure of the access permissions, and therefore be very expensive. However, in this chapter, we show that a transitive closure is not needed for the static case and less expensive analyses can be used to solve this problem in the dynamic cases. More precisely, we have discovered that merely identifying and then restricting a single step data flow, as opposed to the entire path, is sufficient to prevent the unauthorized flow. This new insight has significantly changed the dimensions of the problem and allows us to offer a variety of strategies and solution techniques that fit different situational needs.

Consider the following situations which have differing solution requirements. For example, in embedded system environments complex monitors cannot be deployed due to their computation or power requirements. Similarly, there are solutions for cryptographic access control[16, 20], where accesses are not mediated by a centralized monitor and therefore easily offer distributed trust. In such cases, the access control policy needs to be correct by design. In other situations, when there are no special computational or power constraints, a monitor can be used, and therefore can be utilized to prevent data leakages. However, there may also be situations where access needs to be granted even if a data leakage may occur and then audited after the fact. This would happen in emergencies, which is why break-glass models exist[14, 39, 41]. It also happens, for example, in a hospital environment, where doctors, nurses, and even clerical staff need access to patient records to provide clinical care. While some may abuse their access, it is not feasible to deny access initially since that may hamper patient care. In such cases, the access requests are satisfied, but will likely be audited later to identify problems, which should then be fixed to eliminate future vulnerabilities.

In this chapter, we develop several different solution strategies to address both the confidentiality and integrity violations in DAC and RBAC. Specifically, we propose a conservative approach that analyzes the access control model to identify "potential" unauthorized flows and eliminates them by revoking necessary assignments such as read and write permissions, or permission to role assignments. Since this eliminates all potential unauthorized flows, regardless of whether they actually occur or not, this could be considered too restrictive. However, it is perfectly secure in the sense that no data leakages can ever occur. We also develop a

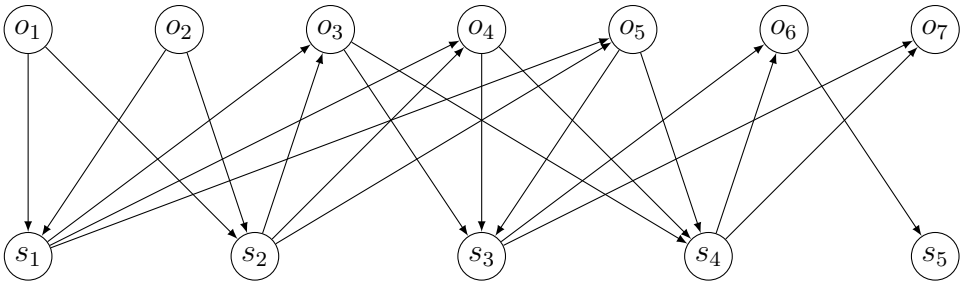


Figure 5.1. Graph representation of the DAC given in Table 5.1

proactive approach, in which object accesses are tracked dynamically at each read and write operation. Thus, any suspicious activity that could lead to an unauthorized data flow can be identified and prevented at the point of time that it occurs. Thus, this approach only restricts access if there is a *signal* for an unauthorized data flow. Finally, we also develop a retrospective approach, that logs all allowed operations and then utilizes the conservative approach as a subroutine to identify and eliminate violations that have occurred and prevent them from occurring again.

Although all three approaches stated above can be applicable to both DAC and RBAC, the way they are implemented slightly differs. In DAC, our application strategies are focused more on utilizing Integer Linear Programming (ILP) models to obtain the optimal solution, whereas the complexity of RBAC forces us to use heuristic based solution strategies instead. We propose these approaches with detailed and customized definitions given in the following two sections.

5.2 Analysis of Data Leakage Problem in DAC

In our analysis for DAC, we make use of the graph representation, defined in Chapter 3. As an example, Figure 5.1 shows the graph representation of the DAC

shown in Table 5.1. Utilizing the graph representation, we first define the flow and vulnerability paths.

Definition 17. (Flow Paths) In an access control system C, a *flow path* from object o to object o', denoted $o \rightsquigarrow o'$, is a path in G_C from o to o', which points out the possibility of copying the content of o into o'.

The *length* of a flow path corresponds to the number of subjects along the path. For example, $o_1 \rightarrow_r s_1 \rightarrow_w o_3$ (denoted as $o_1 \rightsquigarrow o_3$) is a flow path of length 1, while $o_1 \rightarrow_r s_1 \rightarrow_w o_3 \rightarrow_r s_3 \rightarrow_w o_6$ (denoted as $o_1 \rightsquigarrow o_6$) is a flow path of length 2 of the DAC shown in Figure 5.1. In all, there are 12 flow paths of length 1, while there are 4 flow paths of length 2 in the DAC shown in Figure 5.1.

Definition 18. (Confidentiality Vulnerability) A DAC, C has a *confidentiality vulnerability*, if there are two objects o and o', and a subject s such that $o \rightsquigarrow o' \rightarrow_r s$ (*confidentiality vulnerability path* or simply *vulnerability path*), and $o \not\rightarrow_r s$.

A confidentiality vulnerability, shows that subject s (the *violator*) can *potentially* read the content of object o through o', though s is not allowed to read directly from o. We represent confidentiality vulnerabilities using triples of the form (o, o', s). For example, the DAC depicted in Figure 5.1(a) has the confidentiality vulnerability (o_1, o_3, s_3) since $o_1 \rightsquigarrow o_3$ and $o_3 \rightarrow_r s_3$ but $o_1 \not\rightarrow_r s_3$. Similarly, (o_2, o_6, s_5) is another confidentiality vulnerability since $o_2 \rightsquigarrow o_6$ and $o_6 \rightarrow_r s_5$ but $o_2 \not\rightarrow_r s_5$. In total, there are 15 confidentiality vulnerabilities, given below.

$$
\begin{array}{cccc}
c1 : (o_1, o_3, s_3), & c2 : (o_1, o_3, s_4), & c3 : (o_1, o_4, s_3), & c4 : (o_1, o_4, s_4), \\
c5 : (o_1, o_5, s_3), & c6 : (o_1, o_5, s_4), & c7 : (o_2, o_3, s_3), & c8 : (o_2, o_3, s_4), \\
c9 : (o_2, o_4, s_3), & c10 : (o_2, o_4, s_4), & c11 : (o_2, o_5, s_3), & c12 : (o_2, o_5, s_4), \\
c13 : (o_5, o_6, s_5), & c14 : (o_1, o_6, s_5), & c15 : (o_2, o_6, s_5). &
\end{array}
$$

Definition 19. (Integrity Vulnerability) A DAC, C has an *integrity vulnerability*, if there exist a subject s, and two objects o and o' such that $s \rightarrow_w o$, $o \rightsquigarrow o'$ (*integrity vulnerability path* or simply *vulnerability path*) and $s \not\rightarrow_w o'$. An integrity vulnerability, shows that subject s (the *violator*) can indirectly write into o' using the path flow from o to o', though s is not allowed to write directly into o'.

We represent integrity vulnerabilities using triples of the form (s, o, o'). For example, the DAC depicted in Figure 5.1 has the integrity vulnerability (s_1, o_3, o_6) since $o_3 \rightsquigarrow o_6$ and $s_1 \rightarrow_w o_3$ but $s_1 \not\rightarrow_w o_6$. In total, there are 12 integrity vulnerabilities, given below.

$$\begin{array}{llll} i1 : (s_1, o_3, o_6), & i2 : (s_1, o_3, o_7), & i3 : (s_1, o_4, o_6), & i4 : (s_1, o_4, o_7), \\ i5 : (s_1, o_5, o_6), & i6 : (s_1, o_5, o_7), & i7 : (s_2, o_3, o_6), & i8 : (s_2, o_3, o_7), \\ i9 : (s_2, o_4, o_6), & i10 : (s_2, o_4, o_7), & i11 : (s_2, o_5, o_6), & i12 : (s_2, o_5, o_7). \end{array}$$

When a DAC has either a confidentiality or an integrity vulnerability, we simply say that C has a *vulnerability*, whose *length* is that of its underlying vulnerability path. Thus, for the DAC depicted in Figure 5.1, there are $15 + 12 = 27$ vulnerabilities.

A vulnerability in an access control system does not necessarily imply that a data leakage (confidentiality or integrity violation) occurs. Rather, a leakage can potentially happen unless it is detected and blocked beforehand, using for example a monitor. Before we define this notion formally, we first develop the necessary formalism.

Definition 20. A *run* of a DAC, C is any finite sequence $\pi = (s_1, op_1, o_1) \ldots (s_n, op_n, o_n)$ of triples (or *actions*) from the set $S \times \{read, write\} \times O$ such that for every $i \in [1, n]$ one of the following cases holds:

[Read] $op_i = read$, and $o_i \rightarrow_r s_i$;

[Write] $op_i = write$, and $s_i \rightarrow_w o_i$.

A run π represents a sequence of allowed read and write operations executed by subjects on objects. More specifically, at step $i \in [n]$ subject s_i accomplishes the operation op_i on object o_i. Furthermore, s_i has the right to access o_i in the op_i mode.

A run π has a *flow* from an object $\hat{o}_1$ to a subject $\hat{s}_k$ provided there is a flow path $\hat{o}_1 \rightarrow_r \hat{s}_1 \rightarrow_w \hat{o}_2 \ldots \hat{o}_k$ and $\hat{o}_k \rightarrow_r \hat{s}_k$ such that $(\hat{s}_1, read, \hat{o}_1)(\hat{s}_1, write, \hat{o}_2) \ldots (\hat{s}_k, read, \hat{o}_k)$ is a sub-sequence of π. Similarly, we can define flows from subjects to objects, objects to objects, and subjects to subjects.

Definition 21. (Confidentiality Violation) A run π of a DAC, C has a *confidentiality violation*, provided there is a confidentiality vulnerability path from an object o to a subject s and π has a flow from o to s. A DAC $\mathcal{C}$ has a *confidentiality violation* if there is a run of C with a confidentiality violation.

Thus, for example, in the DAC depicted in Figure 5.1, a *confidentiality violation* would occur if there was a sequence $(s_1, read, o_1)(s_1, write, o_3)(s_3, read, o_3)$ which was a sub-sequence of π.

Definition 22. (Integrity Violation) A run π of a DAC C has an *integrity violation*, provided there is an integrity vulnerability path from a subject s to an object o and π has a flow from s to o. A DAC C has an *integrity violation* if there is a run of C with an integrity violation.

As above, in the DAC depicted in Figure 5.1, a *integrity violation* would occur, for example, if there was a sequence $(s_2, write, o_4)(s_3, read, o_4)(s_3, write, o_7)$ which was a sub-sequence of π.

A DAC has a *data leakage* if it has either a confidentiality or an integrity violation.

From the definitions above it is straightforward to see that the following property holds.

Proposition 1. *An access control system is data leakage free if and only if it is vulnerability free.*

The direct consequence of the proposition above suggests that a vulnerability free access control system is data leakage free by design, hence it does not require a monitor to prevent data leakages.

We now prove a simple and fundamental property of DAC that constitutes one of the building blocks for our approaches for checking, preventing, and eliminating vulnerabilities / data leakages as shown later in this chapter.

Theorem 1 (FUNDAMENTAL THEOREM)**.** *Let C be an access control system. C has a vulnerability only if C has a vulnerability of length one.*

Proof. The proof is by contradiction. Assume that $\rho = o_0 \rightarrow_r s_0 \rightarrow_w o_1 \ldots s_{n-1} \rightarrow_w o_n$ is vulnerability path. Without loss of generality, assume that ρ is of minimal length. Thus, n is greater than one by hypothesis.

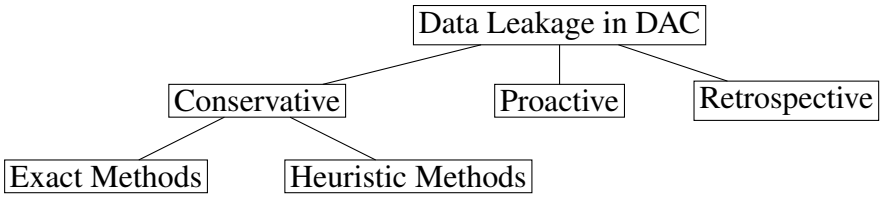


Figure 5.2. Framework for the analysis of data leakage in DAC

We first consider the case of confidentiality vulnerability. Let s be the violator. Since ρ is of minimal length, all objects along ρ except o_0 can be directly read by s (i.e., $o_i \rightarrow_r s$ for every $i \in [1, n]$), otherwise there is an confidentiality vulnerability of smaller length. Thus, $o_0 \rightarrow_r s_0 \rightarrow_w o_1$ is a confidentiality vulnerability of length one, as s can read from o_1 but cannot read from o_0. A contradiction.

We give a similar proof for integrity vulnerabilities. Again, since ρ is of minimal length, all objects along ρ, except o_0, can be directly written by s_0, i.e., $s_0 \rightarrow_w o_i$ for every $i \in \{1, \ldots, n\}$. But, this entails that $o_0 \rightarrow_r s_0 \rightarrow_w o_n$ is an integrity vulnerability of length one (as s can write into o_0 but cannot directly write into o_n). Again, a contradiction. □

The FUNDAMENTAL THEOREM suggests that a data leakage analysis using the transitive closure is not necessary. Instead, an analysis done on the vulnerability paths of length 1 is sufficient. This strong result enables more efficient solution strategies to handle data leakage problem. We now present three alternative strategies for preventing data leakages, which fit different environments, as discussed above (See Figure 5.2.

5.2.1 Exact Methods for Conservative Approach

When a monitor is not possible or even doable the only solution to get an access control that is free of data leakages is that of having the DAC free of vulnerabilities (see Proposition 1). In this section, we propose an automatic approach that turns any DAC into one free of vulnerabilities by revoking permissions.

This can be naively achieved by removing all read and write permissions. However, this would make the whole approach useless.

Instead, it is desirable to minimize the changes to the original access control matrix so as not to disturb the users' ability to perform their job functions, unless it is not absolutely needed. Furthermore, the removal of these permissions should take into account the fact that some of them may belong to *trusted users* (i.e. subjects), such as system administrators, and therefore we want to prevent the removal of these permissions.

We show that this problem is NP-complete. Therefore, an efficient solution is unlikely to exist (unless P=NP).

To circumvent this computational difficulty, we propose a compact encoding of this optimization problem into integer linear programming (ILP) by exploiting Theorem 1. The main goal is that of leveraging efficient solvers for ILP, which nowadays exist. We show this approach is promising in practice in Section 5.2.5.

Definition 23. (Maximal Data Flow Problem) Let $C = \langle S, O, A_R \cup A_W \rangle$ be an access control system, and $T = (A_R^t, A_W^t)$ be the sets of *trusted permissions* where $A_R^t \subseteq A_R$ and $A_W^t \subseteq A_W$. A pair $Sol = (A_R^{sol}, A_W^{sol})$ is a *feasible solution*

of C and T, if $A_R^t \subseteq A_R^{sol} \subseteq A_R$, $A_W^t \subseteq A_W^{sol} \subseteq A_W$ and $C' = \langle S, O, A_R^{sol} \cup A_W^{sol} \rangle$ does not have any vulnerability. The size of a feasible solution Sol, denoted $size(Sol)$, is the value $|A_R^{sol}| + |A_W^{sol}|$.

The *Maximal Data Flow Problem* (MDFP for short) is to maximize $size(Sol)$.

Maximal Data Flow Problem is NP-complete. We first define the decision problem corresponding to MDFP, and then prove that this problem is NP-complete.

Given an instance $I = (C, T)$ of MDFP and a positive integer K, the decision problem associated to MDFP, called D-MDFP, asks whether there is a feasible solution of I of size greater of equal to K. We now show that D-MDFP is NP-complete.

NP-membership. Let $Sol = (A'_R, A'_W)$ such that $A'_R, A'_W \subseteq S \times O$. To check whether Sol is a feasible solution of I, we need to check that (1) $A_R^t \subseteq A'_R \subseteq A_R$, (2) $A_W^t \subseteq A'_W \subseteq A_W$, (3) $|A'_R| + |A'_W| \geq K$, and more importantly, (4) that $\langle S, O, A'_R \cup A'_W \rangle$ is a DAC that does not contain any vulnerability. The first three properties are easy to realize in polynomial time. Concerning the last property, we exploit Theorem 1. To check that there is no confidentiality vulnerability, we build all sequences of the form $o_0 \rightarrow'_r s_0 \rightarrow'_w o_1 \rightarrow'_r s_1$ and then verify the existence of the read permission $o_0 \rightarrow'_r s_1$. Similarly, for integrity vulnerability we build all sequences such that $s_0 \rightarrow'_w o_0 \rightarrow'_r s_0 \rightarrow'_w o_1$ and then check the existence of the write permission $s_0 \rightarrow'_w o_1$. Note that, all these sequences can be built in $O(O^2 \cdot S^2)$ and these checks can all be accomplished in polynomial time. This shows that D-MDFP belongs to NP.

NP-hardness. For the NP-hardness proof, we provide a polynomial time reduction from *the edge deletion transitive digraph* problem (ED-TD for short) to D-MDFP. The ED-TD asks to remove the minimal number of edges from a given directed graph such that the resulting graph corresponds to its transitive closure. ED-TD problem is known to be NP-complete (see [56] Theorem 15, and [57]).

The reduction is as follows. Let $G = (V, E)$ be a directed graph with set of nodes $V = \{1, 2, \ldots n\}$ and set of edges $E \subseteq (V \times V)$. We assume that nodes of G do not have self-loops. We now define the instance $I_G = (C_G, T_G)$ of D-MDFP to which G is reduced to. Let $C_G = \langle S, O, A_R \cup A_W \rangle$ and $T_G = (A_R^t, A_W^t)$. C_G has a subject s_i and an object o_i, for each node $i \in V$. Moreover, there is a read permission from o_i to s_i, and a write permission from s_i to o_i, for every node $i \in V$. These permissions are also trusted, i.e., belonging to A_R^t and A_W^t, respectively; and no further permissions are trusted. Furthermore, for every edge $(i, j) \in E$, there is a read permission from o_i to s_j, and a write permission from s_i to o_j. Formally,

$$S = \{s_i \mid i \in V\} \text{ and } O = \{o_i \mid i \in V\};$$

$$A_R^t = \{(o_i, s_i) \mid i \in V\};$$

$$A_W^t = \{(s_i, o_i) \mid i \in V\};$$

$$A_R = \rightarrow_r^t \cup \{(o_i, s_j) \mid (i, j) \in E\};$$

$$A_W = \rightarrow_w^t \cup \{(s_i, o_j) \mid (i, j) \in E\}.$$

Before we prove that the transformation defined above is a reduction from ED-TD to D-MDFP, we give the following lemma.

Lemma 1. *Let G be a directed graph with nodes $V = \{1, 2, \ldots, n\}$, and $Sol = (A'_R, A'_W)$ be a feasible solution of I_G. For any $i, j \in V$ with $i \neq j$, $o_i \rightarrow'_r s_j$ if and only if $s_i \rightarrow'_w o_j$.*

Proof. The proof is by contradiction. Consider first the case when $o_i \rightarrow'_r s_j$ and $s_i \not\rightarrow'_w o_j$. Observe that, $s_i \rightarrow'_w o_i$ and $s_j \rightarrow_w o_j$ exist as both of them are trusted permissions of I_G. Thus, $s_i \rightarrow'_w o_i \rightarrow'_r s_j \rightarrow_w o_j$ is an integrity vulnerability, leading to a contradiction. The case when $o_i \not\rightarrow'_r s_j$ and $s_i \rightarrow'_w o_j$ is symmetric, and we omit it here. □

We now show that the transformation defined above from G to I_G is indeed a polynomial reduction from ED-TD to D-MDFP. The NP-hardness directly follows from the following lemma.

Lemma 2. *Let G be a directed graph with n nodes. G contains a subgraph G' with K edges whose transitive closure is G' itself if and only I_G admits a feasible solution Sol of size $2 \cdot (n + K)$.*

Proof. Let $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, $G' = (V, E')$, $I_G = (C_G, T_G)$ where $C_G = (S, O, A_R, A_W)$ and $T_G = (A^t_R, A^t_W)$, and $Sol = (A'_R, A'_W)$.

"only if" direction. Assume that G' is the transitive closure of itself and $|E'| = K$. We define Sol as follows:

$$A'_R \;=\; A^t_R \;\cup\; \{o_i \rightarrow_r s_j \mid (i, j) \in E'\}$$

and

$$A'_W \;=\; A^t_W \;\cup\; \{s_i \rightarrow_w o_j \mid (i, j) \in E'\}.$$

From the definition of I_G, it is straightforward to see that $size(Sol) = 2 \cdot (n + K)$. To conclude the proof we only need to show that Sol is a feasible solution of I_G. Since $A_R^t \subseteq A'_R$ and $A_W^t \subseteq A'_W$ we are guaranteed that Sol contains all trusted permissions of T_G. We now show that $C' = (S, O, A_R \cup A_W)$ does not contain any vulnerability. Assume that there is a vulnerability in C'. By Theorem 1, there must be a vulnerability of length one. If it is a confidentiality vulnerability, then $o_i \rightarrow'_r s_k \rightarrow'_w o_z \rightarrow'_r s_j$ and $o_i \not\rightarrow'_r s_j$, for some $i, k, z, j \in V$ with $i \neq j$. From the definition of I_G, it must be the case that there is a path from node i to node j in G' and $(i', j) \notin E$ which leads to a contradiction. The case of integrity vulnerabilities is symmetric.

"if" direction. Assume that Sol is a feasible solution of I_G of size $2 \cdot (n + K)$. We define $E' = \{(i, j) \mid i \neq j \wedge o_i \rightarrow'_r s_j\}$. Note that, in the definition of E' using permission $s_i \rightarrow'_w o_j$ rather than $o_i \rightarrow'_r s_j$ would lead to the same set of edges E' (see Lemma 1). By the definition of I_G and Lemma 1, it is direct to see the G' is a subgraph of G and $|E'| = K$. We now show that the transitive closure of G' is again G'. By contradiction, assume that there is a path from node i to node j in G' and there is no direct edge from i to j. But this implies that in the access control system $(S, O, A'_R \cup A'_W)$ there is a sequence of alternating read and write operations from object o_i to subject s_j and $o_i \not\rightarrow'_r s_j$, which witnesses a confidentiality vulnerability. This is a contradiction as Sol is a feasible solution of I_G. □

We are now ready to claim one of the main results of this section.

Theorem 2. *D-MDFP is NP-complete.*

ILP Formulation. Here we define a reduction from MDFP to integer linear programming (ILP). In the rest of this section, we denote by $I = (C, T)$ to be an instance of MDFP, where $C = (S, O, A_R \cup A_W)$ and $T = (A_R^t, A_W^t)$.

The set of variables $\mathcal{V}$ of the ILP formulation is:

$$\mathcal{V} = \{r_{o,s} \mid o \in O \wedge s \in S \wedge o \rightarrow_r s\} \cup \{w_{s,o} \mid s \in S \wedge o \in O \wedge o \rightarrow_r s\}$$

The domain of the variables in $\mathcal{V}$ is $\{0, 1\}$, and the intended meaning of these variables is the following. Let $\eta_I : \mathcal{V} \rightarrow \{0, 1\}$ be an assignment of the variables in $\mathcal{V}$ corresponding to an optimal solution of the ILP formulation. Then, a solution for I is obtained by removing all permissions corresponding to the variables assigned to 0 by η_I. Formally, $Sol_{\eta_I} = (A_R^{sol}, A_W^{sol})$ is a solution for I, where

$$A_R^{sol} = \{\, (o, s) \mid o \in O \wedge s \in S \wedge o \rightarrow_r s \wedge \eta_I(r_{o,s}) = 1 \,\}$$

and

$$A_W^{sol} = \{\, (s, o) \mid s \in S \wedge o \in O \wedge s \rightarrow_w o \wedge \eta_I(w_{s,o}) = 1 \,\}.$$

The main idea on how we define the ILP encoding, hence its correctness, derives straightforwardly from Theorem 1: we impose that every flow path of length one, say $o \rightarrow_r \hat{s} \rightarrow_w o'$, if these permissions remain in the resulting access control system $C' = (S, O, A_R^{sol}, A_W^{sol})$, then it must be the case that for every subject $s \in S$

- if s can read from o' in C', s must also be able to read from o in C' (CONFIDENTIALITY), and

- if s that can write into o in C', s must be also able to write into o' in C' (INTEGRITY).

Formally, the linear equations of our ILP formulation is the minimal set containing the following.

Confidentiality Constraints: For every sequence of the form $o \rightarrow_r \hat{s} \rightarrow_w \hat{o} \rightarrow_r s$, we add the constraint

$$r_{o,\hat{s}} + w_{\hat{s},\hat{o}} + r_{\hat{o},s} - G \ \leq \ 2$$

where G is $r_{o,s}$ in case $o \rightarrow_r s$, otherwise $G = 0$.

For example, for the sequence $o_1 \rightarrow_r s_1 \rightarrow_w o_3 \rightarrow_r s_2$, in the DAC depicted in Figure 5.1, we add the constraint

$$r_{o_1,s_1} + w_{s_1,o_3} + r_{o_3,s_2} - 0 \ \leq \ 2$$

Integrity Constraints: For every sequence of the form $s \rightarrow_w o \rightarrow_r \hat{s} \rightarrow_w \hat{o}$, we add the constraint

$$w_{s,o} + r_{o,\hat{s}} + w_{\hat{s},\hat{o}} - G \ \leq \ 2$$

where G is $w_{s,\hat{o}}$ in case $s \rightarrow_w \hat{o}$, otherwise $G = 0$.

As above, for the sequence $s_2 \rightarrow_w o_4 \rightarrow_r s_3 \rightarrow_w o_7$, in the DAC depicted in Figure 5.1(a), we add the constraint

$$w_{s_2,o_4} + r_{o_4,s_3} + w_{s_3,o_7} - 0 \ \leq \ 2$$

$$
\begin{aligned}
& max && \sum_{v \in \mathcal{V}} v \\
& subject\ to &&
\end{aligned}
$$

$$
\begin{aligned}
r_{o,\hat{s}} + w_{\hat{s},\hat{o}} + r_{\hat{o},s} - r_{o,s} &\leq 2, \ \ \forall\, o \rightarrow_r \hat{s} \rightarrow_w \hat{o} \rightarrow_r s, o \rightarrow_r s \\
r_{o,\hat{s}} + w_{\hat{s},\hat{o}} + r_{\hat{o},s} &\leq 2, \ \ \forall\, o \rightarrow_r \hat{s} \rightarrow_w \hat{o} \rightarrow_r s, o \not\rightarrow_r s \\
w_{s,\hat{o}} + r_{\hat{o},\hat{s}} + w_{\hat{s},o} - w_{s,o} &\leq 2, \ \ \forall\, s \rightarrow_w \hat{o} \rightarrow_r \hat{s} \rightarrow_w o, s \rightarrow_w o \\
w_{s,\hat{o}} + r_{\hat{o},\hat{s}} + w_{\hat{s},o} &\leq 2, \ \ \forall\, s \rightarrow_w \hat{o} \rightarrow_r \hat{s} \rightarrow_w o, s \not\rightarrow_w o \\
r_{o,s} &= 1, \ \ \forall\, o \rightarrow_r^t s \\
w_{s,o} &= 1, \ \ \forall\, s \rightarrow_w^t o \\
v &\in \{0,1\}, \ \ \forall v \in \mathcal{V}
\end{aligned}
$$

Figure 5.3. ILP formulation of MDFP.

Trusted Read Constraints: For every $o \rightarrow_r^t s$, we have the constraint

$$r_{o,s} = 1.$$

Trusted Write Constraints: For every $s \rightarrow_w^t o$, we have the constraint

$$w_{s,o} = 1.$$

It is easy to see that any variable assignment η that obeys all linear constraints defined above leads to a feasible solution of I.

Objective Function: Now, to maximize the number of left permissions (or equivalently, minimize the number of removed permissions) we define the objective function of the ILP formulation as the sum of all variables in $\mathcal{V}$. Compactly, our ILP-FORMULATION(C, T) is as shown in Figure 5.3.

We now formally state the correctness of our ILP approach, which is entailed from the fact that we remove the minimal number of permissions from $\mathcal{C}$ resulting

in a new DAC that does not have any vulnerability of length one, hence from Theorem 1 does not have any vulnerability at all.

Theorem 3. *For any instance I of MDFP, if η_I is an optimal solution of* ILP-FORMULATION(I) *then Sol_{η_I} is an optimal solution of I.*

It is worth noting that while the composite formulation that includes both confidentiality and integrity constraints (as shown in Figure 5.3) gives the optimal solution, solving two subproblems (one for confidentiality followed by the one for integrity each of which only include the relevant constraints) does not give an optimal solution.

For example, for the DAC depicted in Figure 5.1, if we only eliminate the 15 confidentiality vulnerabilities, the optimal solution is to revoke 5 permissions ($o_1 \rightarrow_r s_1$, $o_2 \rightarrow_r s_1$, $o_1 \rightarrow_r s_2$, $o_2 \rightarrow_r s_2$, and $o_6 \rightarrow_r s_5$). This eliminates all of the confidentiality, while all of the original integrity vulnerabilities still exist. No new vulnerabilities are added. Now, if the integrity vulnerabilities are to be eliminated, the optimal solution is to revoke 4 permissions ($s_3 \rightarrow_w o_6$, $s_3 \rightarrow_w o_7$, $s_4 \rightarrow_w o_6$, $s_4 \rightarrow_w o_7$). Thus, the total number of permissions revoked is 9. However, if both confidentiality and integrity vulnerabilities are eliminated together (using the composite ILP in Figure 5.3), the optimal solution is to simply revoke 6 permissions ($o_3 \rightarrow_r s_3$, $o_4 \rightarrow_r s_3$, $o_5 \rightarrow_r s_3$, $o_3 \rightarrow_r s_4$, $o_4 \rightarrow_r s_4$, $o_5 \rightarrow_r s_4$), which is clearly lower than 9.

Compact ILP Formulation. We now present an improved encoding that extends the ILP formulation described above by merging subjects and objects that have

the same permissions. This allows us to get a much reduced encoding, in terms of variables, with better performances in practice (see Section 5.2.5 for an empirical evaluation).

Equivalent Subjects: For an instance $I = (C, T)$ of MDFP with $C = (S, O, A_R \cup A_W)$ and $T = (A_R^t, A_W^t)$, two subjects are *equivalent* if they have the same permissions. Formally, for a subject $s \in S$, let $read_I(s)$ (respectively, $read_I^t(s)$) denote the set of all objects that can be read (respectively, trust read) by s in C, i.e., $read_I(s) = \{o \in O \mid o \rightarrow_r s\}$ (respectively, $read_I^t(s) = \{o \in O \mid o \rightarrow_r^t s\}$). Similarly, we define $write_I(s) = \{o \in O \mid s \rightarrow_w o\}$ and $write_I^t(s) = \{o \in O \mid s \rightarrow_w^t o\}$. Then, two subjects s_1 and s_2 are *equivalent*, denoted $s_1 \approx s_2$, if

$$\begin{aligned} read_I(s_1) &= read_I(s_2), \\ read_I^t(s_1) &= read_I^t(s_2), \\ write_I(s_1) &= write_I(s_2), \text{ and} \\ write_I^t(s_1) &= write_I^t(s_2). \end{aligned}$$

For every $s \in S$, $[s]$ is the equivalence class of s w.r.t. $\approx$. Moreover, $S^{\approx}$ denotes the quotient set of S by $\approx$. Similarly, we can define the same notion of equivalent objects, with $[o]$ denoting the the equivalence class of $o \in O$, and $O^{\approx}$ denoting the quotient set of O by $\approx$.

Given a read relation $A_R \subseteq O \times S$ and two subjects $s_1, s_2 \in S$, $A_R[s_1/s_2]$ is a new read relation obtained from $\rightarrow_r$ by assigning to s_2 the same permissions that s_1 has in A_R:

$$A_R[s_1/s_2] = \quad (\rightarrow_r \setminus (O \times \{s_2\})) \cup \{(o, s_2) \mid o \in O \ \wedge \ o \rightarrow_r s_1\}.$$

Similarly,

$$A_W[s_1/s_2] = \quad (\rightarrow_w \setminus (\{s_2\} \times O)) \cup \{(s_2, o) \mid o \in O \ \wedge \ s_1 \rightarrow_w o\}.$$

A similar substitution can be defined for objects.

The following lemma states that for any given optimal solution of I it is always possible to derive a new optimal solution in which two equivalent subjects have the same permissions.

Lemma 3. *Let $I = (C, T)$ be an instance of the MDFP problem, s_1 and s_2 be two equivalent subjects of I, and $Sol' = (A_R^{sol}, A_W^{sol})$ be a optimal solution of I. Then, $Sol'' = (A_R^{sol}[s_1/s_2], A_W^{sol}[s_1/s_2])$ is also an optimal solution of I.*

Proof. Assume that S and O are the set of subjects and the set of objects of C, respectively. Let $C' = \langle S, O, A_R^{sol}, A_W^{sol} \rangle$ and $C'' = \langle S, O, A_R^{sol}[s_1/s_2], A_W^{sol}[s_1/s_2] \rangle$.

We first prove (by contradiction) that Sol'' is a **feasible** solution of I. Assume that C'' has a vulnerability. This vulnerability is witnessed by a flow path, say ρ, that must contain s_2. If ρ does not involve s_2 then ρ would also be a vulnerability in C', which cannot be true as Sol' is a feasible solution of I. Now, observe that s_2 can always be replaced by s_1 along any flow path of C'', as s_2 and s_1 have the same neighbor in $G_{C''}$. Thus, the flow path obtained by replacing s_2 with s_1 along ρ, also witnesses a vulnerability in C'. Again a contradiction. Therefore, Sol'' is a feasible solution of I.

We now prove that Sol'' is also **optimal** (that is, $size(Sol') = size(Sol'')$) by showing that s_1 and s_2 have the same number of incident edges in $G_{C'}$. Let n_1

(respectively, n_2) be the number of incident nodes of s_1 (respectively, s_2) in $G_{C'}$. By contradiction, and w.l.o.g., assume that $n_1 > n_2$. Since C'' is obtained from C' by removing first the permissions of s_2 and then adding to s_2 the same permissions of s_1, it must be the case that $size(Sol'') > size(Sol')$. This would entail that Sol' is not an optimal solution, which is a contradiction. □

The following property is a direct consequence of Lemma 3.

Corollary 1. *Let $I = (C, T)$ with $C = \langle S, O, A_R \cup A_W \rangle$ be an instance of the MDFP problem that admits a solution. Then, there exists a solution $Sol = (A_R^{sol}, A_W^{sol})$ of I such that for every pair of equivalent subjects $s_1, s_2 \in S$, s_1 and s_2 have the same permissions in $C = \langle S, O, A_R^{sol} \cup A_W^{sol} \rangle$.*

Lemma 3 and Corollary 1 also hold for equivalent objects. Proofs are similar to those provided above and hence we omit them here.

Compact ILP formulation. Corollary 1 suggests a more compact encoding of the MDFP into ILP. From C, we define a new DAC $C^{\approx}$ by collapsing all subjects and objects into their equivalence classes defined by $\approx$, and by merging permissions consequently (edges of G_C). Formally, $C^{\approx}$ has $S^{\approx}$ as set of subjects and $O^{\approx}$ as set of objects, where the read and write permission sets are defined as follows:

$$A_R^{\approx} = \{ ([o], [s]) \mid o \in O \wedge s \in S \wedge o \rightarrow_r s \},$$

$$A_W^{\approx} = \{ ([o], [s]) \mid s \in S \wedge o \in O \wedge s \rightarrow_w o \}.$$

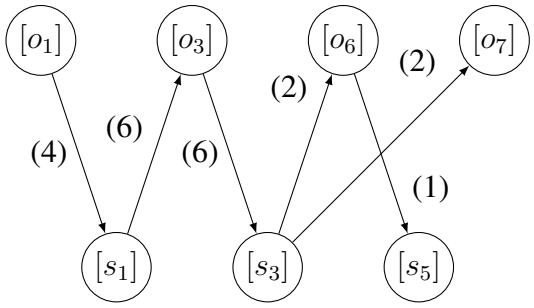


Figure 5.4. Graph representation of the reduced DAC given in Table 5.1

Similarly, we define the trusted permissions of $C^{\approx}$ as $T^{\approx} = (A_R^{t\,\approx}, A_W^{t\,\approx})$ where

$$A_R^{t\,\approx} = \{\ ([o],[s])\ \mid\ o \in O\ \wedge\ s \in S\ \wedge\ o \rightarrow_r^t s\ \},$$
$$A_W^{t\,\approx} = \{\ ([o],[s])\ \mid\ s \in S\ \wedge\ o \in O\ \wedge\ s \rightarrow_w^t o\ \}.$$

We now define a new ILP encoding, COMPACT-ILP-FORMULATION(I), for MFDP on the instance $(C^{\approx}, T^{\approx})$, which is similar to that of Figure 5.3 with the difference that now edges may have a weight greater than one; the weight of an edge will capture the number of edges of C it represents in $C^{\approx}$. More specifically, each edge from a node x_1 to x_2 in $G_{C^{\approx}}$ represents all edges from all nodes in $[x_1]$ to all nodes in $[x_2]$, i.e., its weight is $|[x_1]|\cdot|[x_2]|$. Figure 5.4 represents the compact representation of Figure 5.1, where the edges have the appropriate weights.

Figure 5.5 shows COMPACT-ILP-FORMULATION(I) over the set of variables $\mathcal{V}^{\approx}$. The set of linear constraints is the same as those in Figure 5.3 with the difference that now they are defined over $C^{\approx}$ rather than C. Instead, the objective function is similar to that of Figure 5.3, but now captures the new weighting attributed to edges in $G_{C^{\approx}}$.

Let $\eta_I^{\approx} : \mathcal{V} \rightarrow \{0,1\}$ be a solution to the ILP instance of Figure 5.5. Define

$\widehat{Sol}_{\eta_I^{\approx}} = (\widehat{A_R}^{sol}, \widehat{A_W}^{sol})$ where

$$\widehat{A_R}^{sol} = \{ (o,s) \in O \times S \mid o \rightarrow_r s \ \wedge \ \eta_I^{\approx}(r_{[o],[s]}) \geq 1 \},$$
$$\widehat{A_W}^{sol} = \{ (s,o) \in S \times O \mid s \rightarrow_w o \ \wedge \ \eta_I^{\approx}(w_{[s],[o]}) \geq 1 \}.$$

We now prove that $Sol_{\eta_I^{\approx}}$ is an *optimal* solution of I.

Theorem 4. *For any instance I of MDFP, if $\eta_I^{\approx}$ is an optimal solution of* COMPACT-ILP-FORMULATION(I) *then $\widehat{Sol}_{\eta_I^{\approx}}$ is an optimal solution of I. Furthermore, if I admits a solution then $\eta_I^{\approx}$ also exists.*

Proof. Let $I = (C,T)$, $\widehat{Sol}_{\eta_I^{\approx}} = (\widehat{A_R}^{sol}, \widehat{A_W}^{sol})$, $C' = \langle S, O, \widehat{A_R}^{sol} \cup \widehat{A_W}^{sol} \rangle$, and $C^{\approx} = \langle S^{\approx}, O^{\approx}, A_R^{\approx} \cup A_W^{\approx} \rangle$.

We first show that $\widehat{Sol}_{\eta_I^{\approx}}$ is a **feasible** solution of I. Assume by contradiction that C' has a (one-step, see Theorem 1) confidentiality vulnerability, say

$$o \rightarrow_r^{sol} \hat{s} \rightarrow_w^{sol} o' \rightarrow_r^{sol} s \ \wedge \ o \not\rightarrow_r^{sol} s.$$

It is easy to see that

$$[o] \rightarrow_r^{\approx} [\hat{s}] \rightarrow_w^{\approx} [o'] \rightarrow_r^{\approx} [s] \ \wedge \ [o] \not\rightarrow_r^{\approx} [s]$$

holds, but this is not possible since COMPACT-ILP-FORMULATION(I) contains a constraint that prevents that these relations hold conjunctly. A similar proof exists for integrity vulnerabilities. Therefore, $\widehat{Sol}_{\eta_I^{\approx}}$ is a feasible solution of I.

Now, we show that $\widehat{Sol}_{\eta_I^{\approx}}$ is also *optimal*. Assume by contradiction that $\widehat{Sol}_{\eta_I^{\approx}}$ is not optimal, and $Sol = (A_R^{sol}, A_W^{sol})$ is an optimal solution of I where all equivalent subjects/objects have the same permissions. The existence of Sol is guaranteed by Corollary 1. Now, we reach a contradiction showing that η_I is not optimal

for COMPACT-ILP-FORMULATION(I). For every $s \in S, o \in O$, $\eta(r_{[o],[s]}) = 1$ (respectively, $\eta(w_{[s],[o]}) = 1$) if and only if $o \rightarrow_r^{sol} s$ (respectively, $s \rightarrow_w^{sol} o$) holds. Notice that η is well defined because all subjects/objects in the same equivalent class have the same permissions in Sol. It is straightforward to prove that η allows to satisfy all linear constraints of COMPACT-ILP-FORMULATION(I), and more importantly leads to a greater value of the objective function. Note that, for the variable assignment η the objective function Figure 5.5 has a value $n_\eta = size(Sol)$ whereas has value $n_{\eta_I} = size(\widehat{Sol}_{\eta_I^{\approx}})$ for the assignment $\eta_I^{\approx}$. Now, $n_\eta > n_{\eta_I}$, and it cannot be true because $\eta_I^{\approx}$ is an optimal assignment.

The definition of η and the fact that it satisfy all linear constraints shows that if I admits a solution then it shows that COMPACT-ILP-FORMULATION(I) admit a solution. Therefore, $\eta_I^{\approx}$ also exists. □

$$max \sum_{[o] \rightarrow_r^{\approx} [s]} \left(|\,[o]\,| \cdot |\,[s]\,| \cdot r_{[o],[s]} \right) \quad + \quad \sum_{[s] \rightarrow_w^{\approx} [o]} \left(|\,[s]\,| \cdot |\,[o]\,| \cdot w_{[s],[o]} \right)$$

$$subject\ to$$

$$\begin{aligned}
r_{[o],[\hat{s}]} + w_{[\hat{s}],[\hat{o}]} + r_{[\hat{o}],[s]} - r_{[o],[s]} &\leq 2, \quad \forall \ [o] \rightarrow_r^{\approx} [\hat{s}] \rightarrow_w^{\approx} [\hat{o}] \rightarrow_r^{\approx} [s] \ \wedge \ [o] \rightarrow_r [s] \\
r_{[o],[\hat{s}]} + w_{[\hat{s}],[\hat{o}]} + r_{[\hat{o}],[s]} &\leq 2, \quad \forall \ [o] \rightarrow_r^{\approx} [\hat{s}] \rightarrow_w^{\approx} [\hat{o}] \rightarrow_r^{\approx} [s] \ \wedge \ [o] \not\rightarrow_r [s] \\
w_{[s],[\hat{o}]} + r_{[\hat{o}],[\hat{s}]} + w_{[\hat{s}],[o]} - w_{[s],[o]} &\leq 2, \quad \forall \ [s] \rightarrow_w^{\approx} [\hat{o}] \rightarrow_r^{\approx} [\hat{s}] \rightarrow_w^{\approx} [o] \ \wedge \ [s] \rightarrow_w [o] \\
w_{[s],[\hat{o}]} + r_{[\hat{o}],[\hat{s}]} + w_{[\hat{s}],[o]} &\leq 2, \quad \forall \ [s] \rightarrow_w^{\approx} [\hat{o}] \rightarrow_r^{\approx} [\hat{s}] \rightarrow_w^{\approx} [o] \ \wedge \ [s] \not\rightarrow_w [o] \\
r_{[o],[s]} &= 1, \quad \forall \ [o] \rightarrow_r^{t} [s] \\
w_{[s],[o]} &= 1, \quad \forall \ [s] \rightarrow_w^{t} [o] \\
v &\in \{0,1\}, \ \forall v \in \mathcal{V}^{\approx}
\end{aligned}$$

Figure 5.5. ILP formulation of MDFP based on equivalence classes.

5.2.2 Heuristic Methods for Conservative Approach

Although the Compact ILP formulation given in Section 5.2.1 reduces the problem significantly, there could still be instances where the Compact ILP formulations fails to generate a solution in a reasonable amount of time. In order to address this issue, we provide a fast heuristic method that is composed of two phases: (1) *Identification* of confidentiality vulnerability paths and integrity vulnerability paths and (2) *Elimination* of confidentiality vulnerability paths and integrity vulnerability paths by greedy revocations.

In particular, the identification phase takes the DAC configuration C as an input and outputs the flow paths that could cause a vulnerability. Following Theorem 1, we propose an algorithm that outputs any confidentiality and integrity vulnerability of length one. We define $\mathcal{T}_{\mathcal{C}}$ to be the set of vulnerability paths of length one that could cause a confidentiality violation and $\mathcal{T}_{\mathcal{I}}$ to be the set of vulnerability paths of length one that could cause an integrity violation. A confidentiality vulnerability path of length one $o \rightarrow_r s \rightarrow_w o' \rightarrow_r s'$ where $o \not\rightarrow_r s'$ is denoted as $(o, s, o', s') \in \mathcal{T}_{\mathcal{C}}$ similarly an integrity vulnerability path of length one $s' \rightarrow_w o \rightarrow_r s \rightarrow_w o'$ where $s' \not\rightarrow_w o'$ is denoted as $(s', o, s, o') \in \mathcal{T}_{\mathcal{I}}$. In Algorithm 2, we provide the pseudocode for identifying the confidentiality vulnerability paths. The algorithm for the integrity vulnerability paths is very similar and therefore omitted here.

Upon the identification phase, the sets $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{I}}$ will be populated with the vulnerability paths. In the elimination phase, these vulnerability paths are eliminated by revoking permissions. However, permission revocations could generate

Algorithm 2 Algorithm to Identify Confidentiality Vulnerability Paths of Length One

Input: C
Output: $\mathcal{T}_{\mathcal{C}}$
Construct the Graph Representation of DAC, $G_C = (\{S \cup O\}, E)$
for all Pairs of object nodes $o, o' \in O$ and subjects $s' \in S$ such that there exists a directed path $(o, s')(s', o') \in E$ **do**
  for all Subject node $s \in S$, such that $\exists (o', s) \in E$ **do**
    if $\nexists (o, s) \in E$ **then**
      $\mathcal{T}_{\mathcal{C}} \leftarrow (o, s', o', s)$
    end if
  end for
end for

new confidentiality and integrity vulnerabilities, so the sets $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{I}}$ might become incomplete. Towards this end, we first clarify how a new vulnerability path might be created as DAC is updated.

1. A new $o^* \rightarrow_r s^*$ is introduced into DAC. Then, there is a possibility for a new confidentiality vulnerability path (o^*, s^*, o, s), if $s^* \rightarrow_w o, o \rightarrow_r s$ and $o^* \not\rightarrow_r s$.

2. A new $s^* \rightarrow_w o^*$ is introduced into DAC. Then there is a possibility for a new confidentiality vulnerability path (o, s^*, o^*, s), if $o \rightarrow_r s^*$, $o^* \rightarrow_r s$ and $s \not\rightarrow_r o$.

3. An existing $o^* \rightarrow_r s^*$ is revoked from DAC. Then there is a possibility for a new confidentiality vulnerability path (o^*, s, o, s^*), if $o^* \rightarrow_r s, s \rightarrow_w o, o \rightarrow_r s^*$.

4. A new $s^* \rightarrow_w o^*$ is introduced into DAC. Then, there is a possibility for a new integrity vulnerability path (s^*, o^*, s, o), if $o^* \rightarrow_r s$, $s \rightarrow_w o$ and $s^* \not\rightarrow_w o$.

5. A new $o^* \rightarrow_r s^*$ is introduced into DAC. Then there is a possibility for a new integrity vulnerability path (s, o^*, s^*, o), if $s \rightarrow_w o^*, s^* \rightarrow_w o$ and $s \not\rightarrow_w o$.

6. An existing $s^* \rightarrow_w o^*$ is revoked from DAC. Then there is a possibility for an new integrity vulnerability path (s^*, o, s, o^*), if $s^* \rightarrow_w o, o \rightarrow_r s, s \rightarrow_w o^*$.

On the other hand, revoking $write$ (resp. $read$) permissions does not generate a new confidentiality (resp. integrity) vulnerability other than eliminating the existing vulnerability. More specifically, if a confidentiality vulnerability path (o, s, o', s') is eliminated by revoking $s \rightarrow_w o'$, then this flow is removed and a new unauthorized data flow will not be generated. Thus, there is no need to perform a new identification again. In order to generalize this result, we provide the next theorem.

Theorem 5. *Eliminating an confidentiality (resp. integrity) vulnerability path by revoking* $write$ *(resp.* $read$*) permission does not generate a new confidentiality (resp. integrity) vulnerability path.*

Proof. The proof follows from the definition of vulnerability paths. In order to have a new confidentiality (resp. integrity) vulnerability path, there has to be a subject s that cannot $read$ (resp. $write$ to) an object o (resp. o'). Since revoking $write$ (resp. $read$) permission does not change any previous $read$ (resp. $write$) access, no new confidentiality (resp. integrity) vulnerability path can be generated.

□

Next, we show that performing $write$ permission revocations for confidentiality vulnerabilities (respectively, $read$ permissions for integrity vulnerabilities) will not generate *new* integrity (respectively, confidentiality) vulnerabilities.

Theorem 6. *Eliminating confidentiality vulnerability paths by only revoking write permissions do not create new integrity vulnerability paths.*

Proof. Suppose to the contrary that the statement is false. Then eliminating a confidentiality vulnerability with a $write$ permission could create a new integrity vulnerability. Suppose that there is a confidentiality vulnerability path $o \rightarrow_r s \rightarrow_w o' \rightarrow_r s'$ in a DAC. Eliminating this vulnerability path is by revoking $(s, writeo')$ could create a new integrity vulnerability, if there exists another flow path $s \rightarrow_w o'' \rightarrow_r s'' \rightarrow_w o'$. Clearly, using this path s could write into o' via subject s''. However, $o'' \rightarrow_r s'' \rightarrow_w o'$ implies there has been other confidentiality vulnerabilities in the DAC:

- If $(s, write, o'')$ and s'' cannot read o, then there is already a confidentiality vulnerability $o \rightarrow_r s \rightarrow_w o' \rightarrow_r s'$ and in order to eliminate it, $(s, write, o'')$ must be revoked. Hence $s \rightarrow_w o'' \rightarrow_r s'' \rightarrow_w o'$ will not exist.

- If $(s, write, o'')$ and $(s'', read, o)$, then there is already a confidentiality vulnerability $o \rightarrow_r s'' \rightarrow_w o' \rightarrow_r s'$ and in order to eliminate it $(s'', write, o')$ must be revoked. Hence $s \rightarrow_w o'' \rightarrow_r s'' \rightarrow_w o'$ will not exist.

Hence, eliminating confidentiality vulnerabilities by revoking only $write$ permissions will not generate new integrity vulnerabilities.

□

Corollary 2. *Eliminating integrity vulnerabilities by revoking read permissions will not create new confidentiality vulnerabilities.*

Theorems 5, 6 and Corollary 2, imply a very strong result that there is no need to perform multiple identification phases as the permissions are revoked. When the sets $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{I}}$ are populated once, the elimination phase will eliminate all vulnerability paths, without generating any new vulnerability paths. Hence in the elimination phase, the following actions are taken:

- $\forall(o, s, o', s') \in \mathcal{T}_{\mathcal{C}}$, revoke $s \rightarrow_w o'$
- $\forall(s', o, s, o') \in \mathcal{T}_{\mathcal{I}}$, revoke $o \rightarrow_r s$

Finally, Lemma 3 and Corollary 1 imply that it is possible to run the heuristic approach over the $C^{\approx}$ in order to obtain a data leakage free C. In this case, for any equivalent subject $A_W[s_1/s_2]$ if $s_1 \rightarrow_w o'$ is revoked for $C^{\approx}$, then $s_2 \rightarrow_w o'$ will also be revoked for C. Similarly, for any equivalent subject $A_R[s_1/s_2]$ if $o \rightarrow_r s_1$ is revoked for $C^{\approx}$, then $o \rightarrow_r s_2$ will also be revoked for C. This reduction provides huge benefits as the heuristic will be able to handle large datasets, which otherwise could not be run without any reduction. This is demonstrated in the Experimental Results for DAC section.

5.2.3 Proactive Approach

The conservative strategies are static in the sense that they are utilized only once to fix all inference problems before the system is actually used. While this ensures complete security, it can potentially impact utility since many actions are

revoked due to the potential of leakage, even if in actuality the situation never occurs. Thus, to reduce the restrictiveness, we now propose a proactive strategy that revokes permissions only if the actions that could cause an unauthorized data flow actually happen. In this approach, we continuously monitor read and write actions only on the identified source and destination objects in $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{I}}$ that could potentially cause an unauthorized flow. More specifically, a confidentiality vulnerability (o, so', s'), can be prevented by revoking any one of the $o \rightarrow_r s$, $s \rightarrow_w o'$ or $o' \rightarrow_r s'$. However, we choose to revoke $o' \rightarrow_r s'$, whenever $o \rightarrow_r s$ and $s \rightarrow_w o'$ occur in this order. Similarly an integrity vulnerability (s', o, s, o'), can be prevented by revoking any one of the $o \rightarrow_r s$, $s \rightarrow_w o'$ or $s' \rightarrow_w o$, and we choose to revoke $s \rightarrow_w o'$, whenever $s' \rightarrow_w o$ and $o \rightarrow_r s$ occur in this order. This way, we revoke permissions, only if a suspicious chain of events occur. Hence, when compared to the conservative approach, the number of revoked permissions is expected to be smaller.

As discussed previously in this chapter, revoking $read$ ($write$, resp.) permission, might invalidate $\mathcal{T}_{\mathcal{C}}$ ($\mathcal{T}_{\mathcal{I}}$, resp.) that has been computed in the identification phase using in Algorithm 2. For example, revoking a $read$ permission means a subject that could read an object before, will not be able to read it anymore. This means there is a possibility that this subject and object could create a new confidentiality vulnerability path. Therefore, the set $\mathcal{T}_{\mathcal{C}}$ ($\mathcal{T}_{\mathcal{I}}$, resp.) should be re-populated whenever a $read$ ($write$, resp.) permission is revoked.

The sequence of the events that cause unauthorized data flow is also important. A preventative action is only meaningful if a $read$ operation precedes a $write$ for confidentiality vulnerabilities and a $write$ operation precedes a $read$ operation

for integrity vulnerabilities. Furthermore, we impose a *windowing* approach, stating that, if the succeeding event occurs after a predefined time window of size θ units, then the preceding event is considered to be outdated, so it is assumed that a data flow will not be unauthorized. Setting $\theta = \infty$ will eventually make the access configuration read-only for each destination object involved in a vulnerability path.

The proactive approach begins with the identification algorithm (Algorithm 2) to populate $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{I}}$. Later, the identified source and destination objects for confidentiality vulnerabilities (i.e. for all $(o, s, o', s') \in \mathcal{T}_{\mathcal{C}}$ the objects o and o'), are monitored for read and write operations (o is monitored for $read$, o' is monitored for $write$). If the subject s reads o and writes into o' within θ amount of time, then $o' \rightarrow_r s'$ is revoked. Likewise the identified source objects for integrity vulnerabilities (i.e. for all $(s, o, s', o') \in \mathcal{T}_{\mathcal{I}}$ the object o), are monitored for read and write operations. If the subject s writes into o and subject s' reads o within θ amount of time, then $s' \rightarrow_w o'$ is revoked. The procedure for confidentiality vulnerabilities is given in Algorithm 3, which requires $O(|\mathcal{T}_{\mathcal{C}}||S|)$ time for prevention and $O(|O|^2|S|)$ time for recomputing $\mathcal{T}_{\mathcal{C}}$.

To show the behavior of the proactive approach, consider again the access control system shown in Table 5.1, along with the potential sequence of operations shown in Table 5.2. Table 5.2 shows the behavior of the proactive approach for each operation in the sequence. Note that the proactive approach revokes a total of six permissions (2 each on operations (2), (3), and (5)).

Algorithm 3 Algorithm for Proactive Strategy

1: **Input:** $\mathcal{T}_C$.
2: Set the read file indicator $\mathcal{R}_o^s = 0, \forall s \in S, o \in O$.
3: Set δ as the current timestamp.
4: *Monitor the access control actions done on the objects on vulnerability paths identified in* $(o, s, o', s') \in \mathcal{T}_C$ *- Use (Alg 2)*
5: **if** o is read by s **then**
6: Set $\mathcal{R}_o^s = \delta$.
7: **end if**
8: **if** o' is written by s and $\delta - \mathcal{R}_o^s \leq \theta$ **then**
9: Revoke read permission $o' \rightarrow_r s'$
10: Redo identification algorithm (Alg 2) to recompute $\mathcal{T}_C$.
11: **end if**

Table 5.2. Sample Sequence of Actions and Monitor's Behavior

	User's Operation	Monitor's Action
1	s_1, r, o_1	
2	s_1, w, o_3	Monitor will revoke $o_3 \rightarrow_r s_3$ $o_3 \rightarrow_r s_4$ to remove the confidentiality vulnerabilities
3	s_1, w, o_4	Monitor will revoke $o_4 \rightarrow_r s_3$ $o_4 \rightarrow_r s_4$ to remove the confidentiality vulnerabilities
4	s_2, w, o_5	
5	s_4, r, o_5	Monitor will revoke $s_4 \rightarrow_w o_6$ and $s_4 \rightarrow_w o_7$ to remove the integrity vulnerability
6	s_3, r, o_3	Access denied
7	s_4, w, o_7	Access denied

5.2.4 Retrospective Approach

In the prior sections, we have described a process to create a data leakage free access control matrix by design as well as a proactive approach that implements *on-demand decision making* to readjust the permissions that cause unauthorized flows. Although both of these accurately eliminate any unauthorized flow, there are several situations where neither of this is appropriate. For example, consider a hospital environment, where the access policy gets updated often, and access needs to typically be granted at the time it is requested, though it can be audited afterwards. It might also be cost prohibitive to implement the proactive approach especially in large computer systems with potentially large number of identified source and destination objects, since the additional overhead to the access evaluation request might not be acceptable.

Also, observing a read on the source object followed by a write on the destination object still does not imply that the unauthorized access will actually happen unless the destination object is read by the subject who does not have access to the source object.

To address the above issues, in this section, we propose to eliminate (future) data leakages through auditing. This is a retrospective strategy that allows all requested operations (thus, enabling some unauthorized data flows to also occur), while logging all of the read and write operations. Periodically, the log to date is analyzed to determine whether an unauthorized data flow has actually happened. If so, we revoke the necessary permissions to block the data flow channels so that they will not be utilized for this purpose in the future.

The unauthorized flows found in the log can be straightforwardly removed using the approach developed in Section 5.2.1. Essentially, it is sufficient to realize that a log of actions $\mathcal{L}$ is equivalent to the notion of a run π. Furthermore, each run π creates a projection of the DAC C, denoted C' which includes all of the subjects that have appeared in the run, along with all of the objects for which any appearing subject has either a $read$ or $write$ permission. Now, we can create the appropriate ILP formulation as given in Figure 5.3 (or in Figure 5.5) for C'. The solution to this will optimally remove all of the confidentiality and integrity violations existing in the log $\mathcal{L}$ and ensure that they cannot reoccur.

Consider again, the log of operations given in Table 5.2. In the retrospective approach, when auditing is done, based on the operations, a project of the original access control matrix (depicted in Table 5.1) is built, containing only the requisite subjects and objects. This projection is shown below:

	o_1	o_3	o_4	o_7
s_1	r	w	w	
s_2	r	w	w	
s_3		r	r	w
s_4		r	r	w

When the corresponding ILP is run, the optimal solution is to revoke 4 permissions (for example, revoking $s_1 \rightarrow_w o_3$, $s_1 \rightarrow_w o_4$, $s_2 \rightarrow_w o_3$, and $s_2 \rightarrow_w o_4$), which is better than the 6 revoked by the monitor based approach. However, 2 data leakages have now occurred (s_3 manages to read o_1 and s_2 manages to write to o_7), which would have been prevented by the monitor based approach.

5.2.5 Experimental Results for DAC

We now present the experimental evaluation which demonstrates the performance and restrictiveness of all of the three proposed approaches. We utilize eight real life access control data sets with users and permissions – namely, (1) fire1, (2) fire2, (3) domino, (4) hc, (5) apj, (6) emea, (7) americas small and (8) americas large that are publicly available [18]. Note that these data sets encode a simple access control matrix denoting the ability of a subject to access an object (in any access mode). Thus, these data sets do not have the information regarding which particular permission on the object is granted to the subject. Therefore, we assume for all of the datasets that each assignment represents both a read and a write permission on a distinct object.

For the conservative approaches, we use the reduced access control matrices obtained by collapsing equivalent subjects and objects, as discussed in Section 5.2.1. The number of subjects and objects in the original and reduced matrices are given in Table 5.3. Note that collapsing subjects and objects significantly reduces the sizes of the datasets (on average the dataset is reduced by 93.82%). Here, by size, we mean the product of the number of subjects and objects. Since the number of constraints is linearly proportional to the number of permissions which depends on the number of subjects and objects, a reduction in their size leads to a smaller ILP problem. On the other hand, we use the original datasets for the proactive and retrospective approaches. Note here that due to hardware constraints we can only report the results for first four datasets for the optimal conservative, proactive and retrospective approaches. Since our proposed heuristic is more scalable, we test larger datasets and report the results.

Table 5.3. Dataset Details

Dataset	Name	Original Size		Reduced Size		Percentage
		Subjects	Objects	Subjects	Objects	Reduction
1	fire1	365	709	90	87	96.97 %
2	fire2	325	590	11	11	99.94 %
3	domino	79	231	23	38	95.21 %
4	hc	46	46	18	19	83.84 %
5	apj	2044	1164	564	578	86.29 %
6	emea	35	3046	34	263	91.61 %
7	americas small	3477	1587	259	349	98.36 %
8	americas large	3485	10127	432	1354	98.34 %

Table 5.4. Results for Optimal Data Leakage Free Access Matrix

Data Set	Orig. CPLEX Time (s)	Red. CPLEX Time (s)	Vulnerabilities	# Perm. Init. Assn.	# Perm. Revoked	% Revoked
1	-	2582	34240	63902	14586	22.83 %
2	-	0.225	514	72856	12014	16.49 %
3	8608.15	6.01	3292	1460	421	28.84 %
4	1262.82	0.27	1770	2972	980	32.97 %

We implement all four solution approaches described in this section. For the optimal conservative approach (Section 5.2.1), we create the appropriate ILP model as per Figure 5.5. The ILP model is then executed using IBM CPLEX (v 12.5.1) running through callable libraries within the code. For the heuristic conservative approach (Section 5.2.2), we use the identification algorithm along with the described permission revocations. For the proactive approach (Section 5.2.3), the identification and monitoring algorithm is implemented. For the retrospective approach (Section 5.2.4), a reduced access control matrix is created from the log and the conservative approach is used as a subroutine to identify the permissions to revoke. The algorithms are implemented in C and run on a Windows machine with 16 GB of RAM and Core i7 2.93 GHz processor.

Table 5.4 presents the experimental results for the optimal conservative ap-

Table 5.5. Results for Data Leakage Free Access Matrix with Heuristic

Data Set	Identification Time (s)	Vulnerabilities	# Perm. Init. Assn.	# Perm. Revoked	% Revoked
1	0.05	34240	63902	31295	48.98 %
2	0.004	514	72856	29367	40.30 %
3	0.05	3292	1460	692	47.39 %
4	≈ 0	1770	2972	1360	45.76 %
5	0.145	18458	13682	4913	35.90 %
6	0.115	121656	14440	5282	36.57 %
7	0.79	412058	210410	57237	27.20 %
8	7.959	2464092	370588	127857	34.50 %

proach. The column "Orig. CPLEX Time", shows the time required to run the ILP formulation given in Figure 5.3, while the column "Red. CPLEX Time" gives the time required to run the compact ILP formulation given in Figure 5.5. As can be seen, the effect of collapsing the subjects and objects is enormous. fire1 and fire2 could not be run (CPLEX gave an out of memory error) for the original access control matrix, while the time required for hc and domino was several orders of magnitude more. Since we use the reduced datasets, as discussed above, the column "Vulnerabilities" reflects the number of vulnerabilities in the reduced datasets to be eliminated. The next three columns depict the amount of permission revocation to achieve a data leakage free access matrix. Note that, here we list the number of permissions revoked in the original access control matrix. On average, 25.28 % of the permissions need to be revoked to get an access control system without any data leakages.

In Table 5.5, we provide experimental results for our heuristic approach. Here, the column "Identification Time" shows the time to run Algorithm 2. Note that time to revoke the permissions from the DAC is negligible and hence not given in the table. The heuristic approach clearly broadens the limits of the conservative

approach to much larger datasets which would not run in the exact methods. On the other hand the number of permissions revoked is 45% more in average for the first four datasets. Although this may not be favorable in real world applications, this heuristic is useful to obtain a data leakage free DAC for large datasets that cannot be handled by the ILP model.

When we have a proactive approach, as discussed in Section 5.2.3, revocations can occur on the fly. Therefore, to test the relative performance of this approach, we have randomly generated a set of read/write operations that occur in the order they are generated. The proactive approach is run and the number of permissions revoked is counted. Since the number of flows can increase as more operations occur, and therefore lead to more revocations, we actually count the revocations for a varying number of operations. Specifically, for each dataset, we generate on average 100 operations for every subject (i.e., we generate $100 * |S|$ number of random operations). Thus, for hc, since there are 46 subjects, we generate 4600 random operations, where as for fire1 which has 365 subjects, we generate 36500 random operations. Now, we count the number of permissions revoked if only $10\% * |S|$ operations are carried out (and similarly for $50\% * |S|$, $100\% * |S|$, $1000\% * |S|$, $5000\% * |S|$, and finally $10000\% * |S|$). Table 5.6 gives the results. Again, we list the number of permissions revoked in the original access control matrix. As we can see, the number of permissions revoked is steadily increasing, and in the case of fire1 and hc the final number of permissions revoked is already larger than the permissions revoked in the conservative method. Also, note that in the current set of experiments, we have set a window size of 1000 – this means that if the gap between a subject reading an object and then writing to another

object is more than 1000 operations, then we do not consider a data flow to have occurred (typically a malicious software would read and then write in a short duration of time) – clearly, the choice of 1000 is arbitrary, and in fact, could be entirely removed, to ensure no data leakages. In this case, the number of permission revocations would be even larger than what is reported, thus demonstrating the benefit of the conservative approach when a large number of operations are likely to be carried out.

Table 5.6. Results for Monitor based approach

Data Set	# Perm. Init. Assn.	Number Permissions Revoked						% Finally Revoked
		10%	50%	100%	1000%	5000%	10000%	
1	63902	0	140	532	14221	24031	26378	41.28 %
2	72856	0	13	26	3912	8129	9025	12.39 %
3	1460	0	36	41	130	283	364	24.93 %
4	2972	0	0	0	557	1123	1259	42.36 %

We evaluated the retrospective approach similar to the proactive approach, by using the same set of random operations generated for the monitor based approach. The log after every 2000 read/write operations is audited and the number of permissions revoked is counted. As expected, the retrospective approach revokes less permissions than the monitor based approach. For example, with fire2, after all of the operations have taken place, finally 5518 permissions (7.57%) are revoked in the retrospective approach, where as 9025 permissions (12.39%) are revoked by the monitor based approach. This is due to two reasons – first, in the retrospective approach, the optimal revocations are found using the ILP formulation; secondly, in the proactive approach revocations are done to eliminate the possibility of a violation from taking place, where as in the retrospective approach, the violation has already taken place. Table 5.7 gives the final number of

revoked permissions for all of the real datasets. It is worth noting that for all 4 of the datasets, the retrospective approach revokes significantly less permissions than the proactive approach to prevent future leakages (of course, the proactive approach prevents all leakages which have been already been allowed by the retrospective approach, though no new leakages of the same type will be allowed by either).

Table 5.7. Results for Retrospective Approach

Data Set	# Perm. Init. Assn.	# Perm Revoked	% Finally Revoked
1	63902	18428	28.84 %
2	72856	5518	7.57 %
3	1460	159	10.89 %
4	2972	980	32.97 %

5.3 Analysis of Data Leakage Problem in RBAC

RBAC is also prone to unauthorized flows just like DAC, and preventative measures are required to eliminate any such flow. On the one hand, the presence of roles makes RBAC standout in administrative efficiency. On the other hand, they complicate the unauthorized flow detection and prevention. Although, similar strategies can be utilized, it is vital to determine whether a role to be revoked from a user, or a permission to be revoked from a role. Due to the complexity of the RBAC model, we cannot utilize ILP models as extensive as we use them in DAC. Therefore, our methodologies primarily depend on heuristic algorithms that provide different levels of restrictiveness.

In order to stay consistent throughout this chapter, we call users as subjects, and permissions as objects for RBAC. Hence, without loss of generality we as-

sume the following: S is a finite set of *subjects*, O is a finite set of *objects*, $SA \subseteq S \times R$ as Subject-Role assignment, $OA = OA_R \cup OA_W$ is the Object-Role assignment where $OA_R \subseteq O \times R$ and $OA_W \subseteq R \times O$. A pair $(s, r) \in SA$, also denoted $s \leftrightarrow r$, is a subject-role assignment representing that subject s is assigned *role* r. A pair $(o, r) \in OA_R$, denoted $o \rightarrow_r r$, is an assignment representing that object o can be readable using role r, whereas $(r, o) \in OA_W$ denoted by $r \rightarrow_w o$, is an assignment representing that object o can be writable using role r.

Similar to DAC, RBAC can also be represented by graphs. We define the graph representation of RBAC as follows:

Definition 24. (Graph Representation of RBAC) The *graph of an RBAC*, $C_R = \langle S, O, R, SA, OA \rangle$, denoted G_{C_R}, is the tripartite graph $(S \cup O \cup R, SA \cup OA)$ whose partition has the parts S, R, and O,where the set SA denotes the edges between S and R and OA denotes the edges between R and O.

Figure 5.6 shows the graph representation of the RBAC shown in Tables 5.8 and 5.9.

Table 5.8. Example Subject-Role Assignment

	r_1	r_2	r_3	r_4
s_1		1		
s_2	1	1		
s_3			1	
s_4			1	
s_5				1

Table 5.9. Example Object-Role Assignment

	r_1	r_2	r_3	r_4
o_1	r			
o_2	r			
o_3		w	r	
o_4		w	r	
o_5		w		
o_6			w	r
o_7			w	

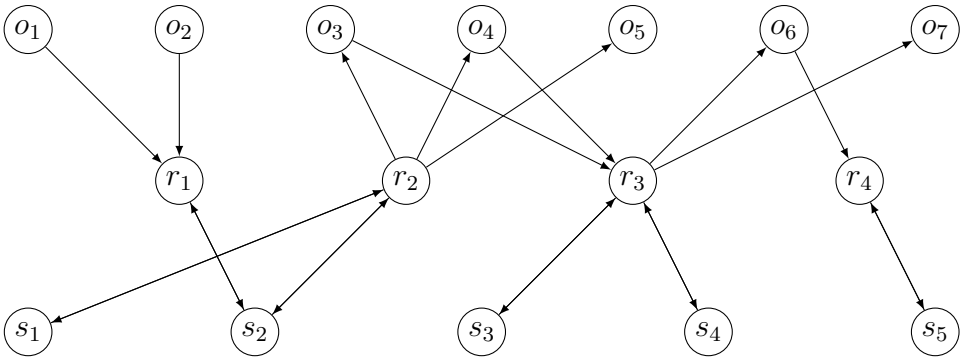


Figure 5.6. Graph representation of RBAC configuration given in Tables 5.8 and 5.9

Definition 25. (Vulnerability Paths in RBAC) In an RBAC, C_R, a *flow path* from object o to object o', denoted $o \rightsquigarrow o'$, is a path in G_{C_R} from o to o' via a subject s and roles r', r'', which points out the possibility of copying the content of o into o'.

The *length* of a flow path corresponds to the number of subjects along the path. For example, $o_1 \rightarrow_r r_1 \rightarrow s_1 \rightarrow r_2 \rightarrow_w o_3$ (denoted as $o_1 \rightsquigarrow o_3$) is a flow path of length 1, while $o_1 \rightarrow_r r_1 \rightarrow s_1 \rightarrow r_2 \rightarrow_w o_3 \rightarrow_r r_3 \rightarrow s_3 \rightarrow r_3 \rightarrow_w o_6$ (denoted as $o_1 \rightsquigarrow o_6$) is a flow path of length 2 of the RBAC shown in Figure 5.6. In all, there are 12 flow paths of length 1, while there are 4 flow paths of length 2 in the RBAC shown in Figure 5.6.

Definition 26. (Confidentiality Vulnerability in RBAC) An RBAC, C_R has a *confidentiality vulnerability*, if there are two objects o and o', a subject s and a role r such that $o \rightsquigarrow o' \rightarrow_r r \leftrightarrow s$ (*confidentiality vulnerability path* or simply *vulnerability path*), and either $\not\exists r'$ such that $s \leftrightarrow r'$ and $o \rightarrow_r r'$ or $\forall r' \leftrightarrow s$ $o \not\rightarrow_r r'$ $o \not\rightarrow_r s$. A confidentiality vulnerability, shows that subject s (the *violator*) can *potentially* read the content of object o through o', though s is not allowed to

read directly from o. We represent confidentiality vulnerabilities using quadruples of the form (o, o', r, s).

For example, the RBAC depicted in Figure 5.6 has 10 confidentiality vulnerabilities, given below.

$$c1 : (o_1, o_3, r_3, s_3),\ c2 : (o_1, o_3, r_3, s_4),\ c3 : (o_1, o_4, r_3, s_3),$$
$$c4 : (o_1, o_4, r_3, s_4),\ c5 : (o_2, o_3, r_3, s_3),\ c6 : (o_2, o_3, r_3, s_4),$$
$$c7 : (o_2, o_4, r_3, s_3),\ c8 : (o_2, o_4, r_3, s_4),\ c9 : (o_1, o_6, r_4, s_5),$$
$$c10 : (o_2, o_6, r_4, s_5).$$

Definition 27. (Integrity Vulnerability) An RBAC, C_R has an *integrity vulnerability*, if there exist a subject s, a role r and two objects o and o' such that $s \leftrightarrow r$, $r \rightarrow_w o$, $o \rightsquigarrow o'$ (*integrity vulnerability path* or simply *vulnerability path*) and either $\not\exists r'$ such that $s \leftrightarrow r'$ and $r' \rightarrow_w o'$ or $\forall r' \leftrightarrow s$, $r' \not\rightarrow_w o$. An integrity vulnerability, shows that subject s (the *violator*) can indirectly write into o' using the path flow from o to o', though s is not allowed to write directly into o'. We represent integrity vulnerabilities using quadruples of the form (s, r, o, o').

For example, the RBAC depicted in Figure 5.6 has 8 integrity vulnerabilities, given below.

$$i1 : (s_1, r_2, o_3, o_6),\ i2 : (s_1, r_2, o_3, o_7),\ i3 : (s_1, r_2, o_4, o_6),$$
$$i4 : (s_1, r_2, o_4, o_7),\ i5 : (s_2, r_2, o_3, o_6),\ i6 : (s_2, r_2, o_3, o_7),$$
$$i7 : (s_2, r_2, o_4, o_6),\ i8 : (s_2, r_2, o_4, o_7),$$

When an RBAC has either a confidentiality or an integrity vulnerability, we simply say that C_R has a *vulnerability*, whose *length* is that of its underlying vulnerability path. Thus, for the RBAC depicted in Figure 5.6, there are $10 + 8 = 18$ vulnerabilities.

Data Leakages in RBAC. As it is similar to DAC, a vulnerability in an RBAC also does not necessarily imply that a data leakage (confidentiality or integrity violation) occurs. A *run* of an RBAC C_R is any finite sequence $\pi = (s_1, r_1, op_1, o_1) \dots (s_n, r_n, op_n, o_n)$ of quadruples (or *actions*) from the set $S \times R \times \{read, write\} \times O$ such that for every $i \in [1, n]$ one of the following cases holds:

[Read] $op_i = read$, and $o_i \rightarrow_r r_i \leftrightarrow s_i$;

[Write] $op_i = write$, and $s_i \rightarrow_w \leftrightarrow o_i$.

A run π represents a sequence of allowed read and write operations executed by subjects on objects. More specifically, at step $i \in [n]$ subject s_i accomplishes the operation op_i on object o_i via role r_i. Furthermore, r_i has the right to access o_i in the op_i mode.

A run π has a *flow* from an object $\hat{o}_1$ to a subject $\hat{s}_k$ provided there is a flow path $\hat{o}_1 \rightarrow_r \hat{r}_1 \rightarrow \hat{s}_1 \rightarrow \hat{r}_2 \rightarrow_w \hat{o}_2 \dots \hat{o}_k$ and $\hat{o}_k \rightarrow_r \hat{r}_k \rightarrow \hat{s}_k$ such that $(\hat{s}_1, \hat{r}_1, read, \hat{o}_1)(\hat{s}_1, \hat{r}_2, write, \hat{o}_2) \dots (\hat{s}_k, \hat{r}_k, read, \hat{o}_k)$ is a sub-sequence of π. Similarly, we can define flows from subjects to objects, objects to objects, and subjects to subjects.

Definition 28. (Confidentiality Violation in RBAC) A run π of an RBAC C_R has a *confidentiality violation*, provided there is a confidentiality vulnerability path from an object o to a subject s and π has a flow from o to s. An RBAC C_R has a *confidentiality violation* if there is a run of C_R with a confidentiality violation.

Thus, for example, in the RBAC depicted in Figure 5.6, a *confidentiality violation* would occur if there was a sequence $(s_1, r_1, read, o_1)(s_1, r_2, write, o_3)(s_3, r_3, read, o_3)$

which was a sub-sequence of π.

Definition 29. (Integrity Violation in RBAC) A run π of an RBAC C_R has an *integrity violation*, provided there is an integrity vulnerability path from a subject s to an object o and π has a flow from s to o. An RBAC C_R has an *integrity violation* if there is a run of C_R with an integrity violation.

As above, in the RBAC depicted in Figure 5.6, a *integrity violation* would occur, for example, if there was a sequence $(s_2, r_2, write, o_4)(s_3, r_3, read, o_4)(s_3, r_3, write, o_7)$ which was a sub-sequence of π.

An RBAC has a *data leakage* if it has either a confidentiality or an integrity violation.

Using the definitions above, the following proposition holds for RBAC models.

Proposition 2. *An RBAC is data leakage free if and only if it is vulnerability free.*

The direct consequence of the proposition above suggests that a vulnerability free access control system is data leakage free by design, hence it does not require a monitor to prevent data leakages.

The Fundamental Theorem (Theorem 1) also applies to RBAC data leakages. Here we provide the modified version of the theorem and its proof tailored specifically for RBAC model.

Theorem 7 (FUNDAMENTAL THEOREM FOR RBAC)**.** *Let C_R be an RBAC configuration. C_R has a vulnerability only if C_R has a vulnerability of length one.*

Proof. The proof is by contradiction. Assume that $\rho = o_0 \rightarrow_r r_0 \rightarrow s_0 \rightarrow r_1 \rightarrow_w o_1 \ldots s_{n-1} \rightarrow r_n \rightarrow_w o_n$ is vulnerability path. Without loss of generality, assume that ρ is of minimal length. Thus, n is greater than one by hypothesis.

We first consider the case of confidentiality vulnerability. Let s be the violator. Since ρ is of minimal length, all objects along ρ except o_0 can be directly read by s (i.e., $o_i \rightarrow_r r_j \rightarrow s$ for every $i \in [1, n]$ and corresponding $j \in [1, m]$), otherwise there is an confidentiality vulnerability of smaller length. Thus, $o_0 \rightarrow_r r_0 \rightarrow s_0 \rightarrow r_1 \rightarrow_w o_1$ is a confidentiality vulnerability of length one, as s can read from o_1 but cannot read from o_0. we reach a contradiction.

We give a similar proof for integrity vulnerabilities. Again, since ρ is of minimal length, all objects along ρ, except o_0, can be directly written by s_0, i.e., $s_0 \rightarrow r_j \rightarrow_w o_i$ for every $i \in [1, n]$ and corresponding $j \in [1, m]$. But, this entails that $o_0 \rightarrow_r r_1 \rightarrow s_0 \rightarrow r_n \rightarrow_w o_n$ is an integrity vulnerability of length one (as s can write into o_0 but cannot directly write into o_n). We reach a contradiction again. □

As a result of the Fundamental Theorem for RBAC, we can provide similar data leakage identification and elimination approaches that utilize only vulnerabilities of length one. In the next sections, we discuss these approaches. Although the main idea behind the solution strategies are similar, the way we handle them differs due to the complexity of the RBAC model. Figure 5.7 outlines the framework we use to address data leakage problem in RBAC.

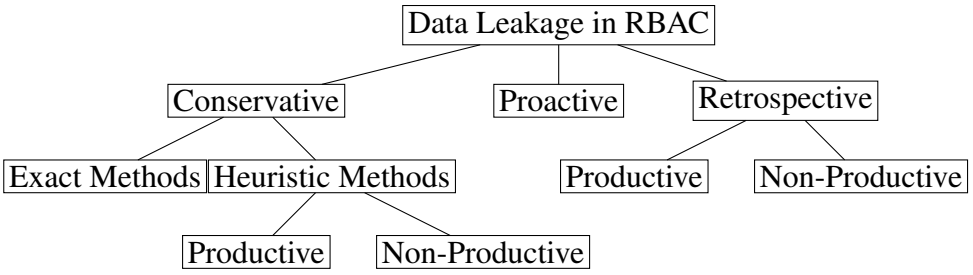


Figure 5.7. Framework of the analysis of data leakage problem in RBAC

5.3.1 Exact Methods for Conservative Approach

A data leakage free RBAC configuration is a more challenging task than designing a DAC configuration. In RBAC, not only is there an additional layer of complexity – the roles – to consider during the design, but also it is crucial to determine which relation – SA or OA or both – to modify. There could be many different ways of achieving a data leakage free RBAC configuration, though serious emphasis must be made on any unnecessary revocations to non-vulnerability paths. In particular, revoking a role from a user implicitly revokes one or more permissions (object accesses). For instance, given a role, if there is only a single permission, that grants an object access, is identified as a part of a vulnerability path, revoking this role will not only eliminate this vulnerability path, but also will eliminate many other paths that may or may not be identified as a vulnerability path. In contrast, if the permission is revoked from the role, then the non-vulnerability paths will not be affected. However, the users who are not on this vulnerability path will unnecessarily lose an object access.

For example, consider the RBAC configuration given in Figure 5.6. In this configuration there is a confidentiality vulnerability (o_1, o_3, r_3, s_3). This confidentiality vulnerability can be eliminated by revoking $r_2 \rightarrow_w o_3$. However, s_1 will

lose write access to o_3 even if s_1 is not on the vulnerability path.

In order to address this problem, we develop a mathematical representation that not only minimizes the number of user-role and role-permission assignments revoked, but also it minimizes the number of assignments revoked unnecessarily. First, we define the decision variables. In the model, there are three sets of decision variables:

$$\begin{aligned}\mathcal{V}^{\mathcal{R}} \;=\; & \{sa_{s,r} \mid s \in S \wedge r \in R \wedge s \leftrightarrow r\} \cup \{oar_{r,o} \mid r \in R \wedge o \in O \wedge o \rightarrow_r r\} \cup \\ & \{oaw_{r,o} \mid r \in R \wedge o \in O \wedge r \rightarrow_w o\} \end{aligned} \tag{5.1}$$

Since the objective of the model is slightly different than that of the ILP formulation for DAC, a simple sum of decision variables, as in the case of the DAC, does not reflect the number of revoked object accesses. Therefore, the objective function is defined as follows:

$$Z = \sum_{\forall s \leftrightarrow r, r \rightarrow_w o, o \rightarrow_r w} sa_{s,r} oar_{r,o} + sa_{u,r} oaw_{r,o}$$

The multiplication of sa and oar-oaw variables in the objective function counts in the effect of revoking multiple object accesses by revoking a role from a user. Now, we give the mathematical formulation.

The structure of the constrains are similar to the ILP model given in Figure 5.3. The first two constraints are to eliminate confidentiality vulnerabilities and the latter two is for integrity vulnerabilities. Due to the complexity of the RBAC

$$
\begin{aligned}
&max && Z \\
&subject\ to &&
\end{aligned}
$$

$$
\begin{aligned}
sa_{s,r'} + oar_{r',o} + sa_{s,r''} + oaw_{r'',\hat{o}} + \quad & \\
sa_{\hat{s},r'''} + oar_{r''',\hat{o}} - (sa_{\hat{s},r''''} oar_{r'''',o}) \quad & \leq 5, \quad \forall\, s \leftrightarrow r', o \rightarrow_r r', s \leftrightarrow r'', \\
& \quad r'' \rightarrow_w \hat{o}, \hat{s} \leftrightarrow r''', \hat{o} \rightarrow_r r''', \hat{s} \leftrightarrow r'''', o \rightarrow_r r'''' \\
sa_{s,r'} + oar_{r',o} + sa_{s,r''} + oaw_{r'',\hat{o}} + \quad & \\
sa_{\hat{s},r'''} + oar_{r''',\hat{o}} \quad & \leq 5, \quad \forall\, s \leftrightarrow r', o \rightarrow_r r', s \leftrightarrow r'', \\
& \quad r'' \rightarrow_w \hat{o}, \hat{s} \leftrightarrow r'''\hat{o} \rightarrow_r r''' \\
sa_{\hat{s},r'} + oaw_{r',o} + sa_{s,r''} + oar_{r'',o} + \quad & \\
sa_{s,r'''} + oaw_{r''',\hat{o}} - (sa_{\hat{s},r''''} oaw_{r'''',\hat{o}}) \quad & \leq 5, \quad \forall\, \hat{s} \leftrightarrow r', r' \rightarrow_w o, s \leftrightarrow r'', \\
& \quad o \rightarrow_r r'', s \leftrightarrow r''', r''' \rightarrow_w \hat{o}, \hat{s} \leftrightarrow r'''', r'''' \rightarrow_w \hat{o} \\
sa_{\hat{s},r'} + oaw_{r',o} + sa_{s,r''} + oar_{r'',o} + \quad & \\
sa_{s,r'''} + oaw_{r''',\hat{o}} \quad & \leq 5, \quad \forall\, \hat{s} \leftrightarrow r', r' \rightarrow_w o, s \leftrightarrow r'', \\
& \quad o \rightarrow_r r'', s \leftrightarrow r''', r''' \rightarrow_w \hat{o} \\
sa_{s,r}, oaw_{r,o}, oar_{r,o} \quad & \in \{0,1\}, \quad \forall s \leftrightarrow r, r \rightarrow_w o, o \rightarrow_r r
\end{aligned}
$$

model structure, a confidentiality vulnerability can be eliminated by either revoking the role r' that the subject s uses to read from the source object, or revoking the read permission to that object from r', or the role r'' that the subject s uses to write to the destination object, or revoking the write permission to that object from r'', or revoking role r''' that the subject $\hat{s}$ reads from the destination object, or revoking the read permission to that object from role r'''. At least one of these conditions must be satisfied unless the subject $\hat{s}$ is assigned to a role r'''' that has a read permission to the source object. The integrity constraints have a similar structure.

Although this model will eliminate the vulnerability paths completely in an optimal way to generate a data leakage free RBAC configuration, it is clear that the model is non-linear due to multiplication of decision variables in the objective function and also in the constraints. Although they are decidable, solving non-linear mathematical programs is a significantly challenging task to tackle, hence

we provide an alternative procedure to alleviate this issue.

Proposed Procedure: Our proposed procedure that will provide a data leakage free RBAC configuration comprises of (1) converting the RBAC into DAC, (2) utilizing the ILP formulation for DAC, and (3) using a Role Mining approach to decompose the DAC back into an RBAC configuration. This approach will optimally eliminate the vulnerability paths in the cost of creating an RBAC configuration with a possibly different set of roles.

5.3.2 Heuristic Methods for Conservative Approach

We propose a heuristic approach that provides a data leakage free RBAC configuration. Our proposed heuristic is a two phase procedure: (1) Identification of confidentiality and integrity vulnerability paths and (2) Elimination of confidentiality and integrity vulnerabilities. Although this procedure seems alike to the heuristic approach that we propose for DAC, additional challenges have to be overcome. Particularly, the way that the vulnerabilities are eliminated are tailored specifically for RBAC model to address the issue of whether to revoke a user from a role or revoke a permission from a role.

In the first phase, we present our identification algorithm for confidentiality and integrity vulnerabilities. We define $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ to be the set of vulnerability paths of length one that could cause a confidentiality violation and $\mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$ to be the set of vulnerability paths of length one that could cause an integrity violation. A confidentiality vulnerability path of length one $o \rightarrow_r r' \rightarrow s \rightarrow r'' \rightarrow_w o' \rightarrow_r r''' \rightarrow s'$ where s' cannot read o is denoted as $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ similarly an integrity vulnerability path of length one $s' \rightarrow r''' \rightarrow_w o \rightarrow_r r' \rightarrow s \rightarrow r'' \rightarrow_w o'$

Algorithm 4 Algorithm to Identify Confidentiality Vulnerability Paths of Length One in RBAC

Input: RBAC
Output: $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$
Construct the Graph Representation of RBAC, $G_{C_R} = (\{S \cup R \cup O\}, SA \cup OA)$
for all Object node pairs $o, o' \in O$, roles $r', r'' \in R$, subjects $s \in S$ such that there exists a directed path $(o, r'), (r', s), (s, r''), (r'', o')$ **do**
  for all Subject node s', such that $\exists (o', r'''), (s', r''') \in E$ **do**
    if $\nexists r''''$ such that $(s', r''''), (o, r'''') \in E$ **then**
      $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}} \leftarrow (o, r', s, r'', o', r''', s')$
    end if
  end for
  for all Subject node s', such that $\exists (r''', o), (s', r''') \in E$ **do**
    if $\nexists r''''$ such that $(s', r''''), (r'''', o') \in E$ **then**
      $\mathcal{T}_{\mathcal{I}}^{\mathcal{R}} \leftarrow (s', r''', o, r', s, r'', o')$
    end if
  end for
end for

where s' cannot write into o' is denoted as $(s', r''', o, r', s, r'', o') \in \mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$. In Algorithm 4, we provide the pseudocode for identifying the confidentiality and integrity vulnerability paths to populate sets $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ and $\mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$.

Next, the vulnerabilities in $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ and $\mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$ are eliminated in order to obtain a data leakage free configuration. Theorems 5, 6 and Corollary 2, implies the following elimination strategy for RBAC:

- $\forall (o, r', s, r'', o', r''', s') \in T_C^R$, eliminate the path $s \rightarrow r'' \rightarrow_w o'$
- $\forall (s', r''', o, r', s, r'', o') \in T_I^R$, eliminate the path $o \rightarrow_r r' \rightarrow s$

Here, eliminating paths $s \rightarrow r'' \rightarrow_w o'$ for confidentiality vulnerabilities and $o \rightarrow_r r' \rightarrow s$ for integrity vulnerabilities require more attention as though revoking $s \rightarrow r''$ or $r' \rightarrow s$ may implicitly revoke some permissions that could

yield new confidentiality vulnerabilities or integrity vulnerabilities. For instance, while eliminating the confidentiality vulnerabilities if a role is revoked, then some $read$ permissions that are assigned to this role will also be revoked alongside the $write$ permission that is intended to be revoked. Hence, this could generate new confidentiality vulnerabilities, which require identification algorithm to be run multiple times. Towards this end, we present two elimination strategies that assure no $read$ permission is revoked for confidentiality vulnerabilities and no $write$ permission is revoked for integrity vulnerabilities. Therefore, a single run of the identification algorithm is guaranteed only if the following elimination strategies are implemented.

- **Productive Elimination:** Productive elimination focuses on splitting the existing roles such that users will not lose access to any object that is not on a vulnerability path. More specifically a confidentiality vulnerability $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$, is eliminated by splitting the role r'' into two roles r'' and $r''^{(C)}$ with the following assignments: $r'' \not\rightarrow_w o'$, $r''^{(C)} \rightarrow_w o'$, $\forall \hat{s} \in S \backslash \{s\}$ such that $\hat{s} \leftrightarrow r''$, assign $\hat{s} \leftrightarrow r''^{(C)}$. Likewise an integrity vulnerability $(s', r', o, r'', s, r''', o') \in \mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$, is eliminated by splitting the role r'' into two roles r'' and $r''^{(C)}$ so that $o \not\rightarrow_r r''$, $o \rightarrow_r r''^{(C)}$, $\forall \hat{s} \in S \backslash \{s'\}$ such that $\hat{s} \leftrightarrow r''$, assign $\hat{s} \leftrightarrow r''^{(C)}$. At the end of the splitting, a role merging process traces the newly generated roles and merges the ones that are assigned to the same set of users.

- **Non-Productive Elimination:** Non-productive elimination performs revocations only on OA in order to preserve the number of roles. More specif-

ically a confidentiality vulnerability $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$, is eliminated by revoking $r'' \rightarrow_w o'$ and an integrity vulnerability $(s', r', o, r'', s, r''', o') \in \mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$, is eliminated by revoking $o \rightarrow_r r''$.

Both of these elimination strategies guarantee that no new vulnerability paths will be generated as a result of the actions taken. Note that there is one common case that must be handled the same for both of the strategies stated above. If a role itself is capable of performing the flow (i.e. $o \rightarrow_r r$ and $r \rightarrow_w o'$) then it means any user who is assigned to this role is capable of performing the flow without requiring assignment to any other role. In this case, role splitting does not make sense, since the newly generated role will not have any users assigned. Hence, for any confidentiality vulnerability $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$, where $r' = r''$, $r'' \rightarrow_w o'$ is revoked. Similarly for any integrity vulnerability $(s', r', o, r'', s, r''', o') \in \mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$, where $r'' = r'''$, $o \rightarrow_r r''$ is revoked. Now, we give the pseudocodes of the Productive (Algorithm 5) Non-productive (Algorithm 6) Elimination Strategies for confidentiality vulnerabilities. The algorithms for integrity vulnerabilities are similar and therefore omitted here.

For example consider the RBAC configuration given in Figure 5.6. Concerning only the confidentiality vulnerabilities, the productive elimination strategy will output the configuration given in Figure 5.3.2, whereas the non productive elimination strategy will output the configuration given in Figure 5.3.2. In the productive elimination, we observe that r_2 is split into two roles in order to preserve s_1's write access to objects o_3 and o_4. Therefore, the number of roles increase by 1. On the other hand, s_1 loses its access to these objects in the non-productive

Algorithm 5 Heuristic Algorithm for Productive Elimination

Input: RBAC Configuration and T_C^R
Output: Data Leakage Free RBAC Configuration
for all $(o, r', s, r'', o', r''', s') \in T_C^R$ **do**
  if $r' = r''$ **then**
    Revoke $r'' \rightarrow_w o'$
  end if
  if $r' \neq r''$ **then**
    if $\nexists r \in R$ such that $r \rightarrow_w o'$ is the only assigned permission **then**
      Create a new role, $r^* \in R$ and set $r^* \rightarrow_w o'$.
    end if
    Mark $r'' \rightarrow_w o'$ to be revoked from r''.
    Mark s not to be assigned to r^*.
  end if
end for
for all $r \in R$ **do**
  if There is a permission $r \rightarrow_w o$ marked to be revoked **then**
    Revoke $r \rightarrow_w o$
    Locate the role r^* where $r^* \rightarrow_w o$ is the only assigned permission
    for all Subjects s not marked to be assigned to r^* **do**
      Set $s \leftrightarrow r^*$
    end for
  end if
end for
for all $r_1, r_2 \in R$ **do**
  if $\nexists u$ such that ($s \leftrightarrow r_1$ and $s \not\leftrightarrow r_2$) or ($s \not\leftrightarrow r_1$ and $s \leftrightarrow r_2$) **then**
    Copy the permission assigned to r_2
    Delete role r_2
  end if
end for

Algorithm 6 Heuristic Algorithm for Non-Productive Elimination

Input: RBAC Configuration and T_C^R
Output: Data Leakage Free RBAC Configuration
for all $(o, r', s, r'', o', r''', s') \in T_C^R$ **do**
  Revoke $r'' \rightarrow_w o'$
end for

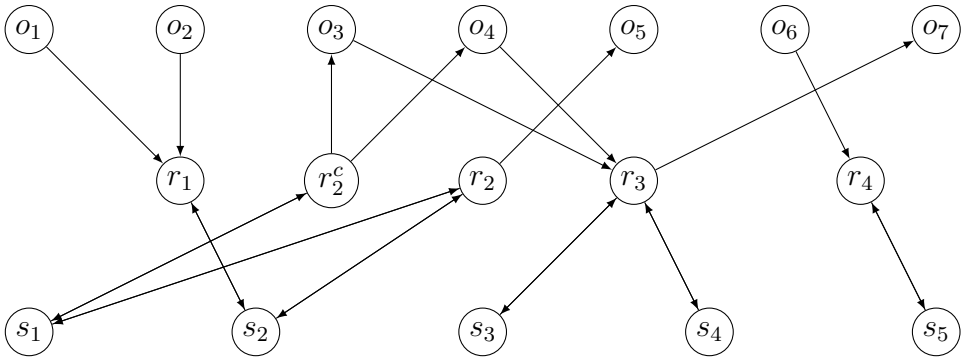


Figure 5.8. The output of the productive elimination strategy performed on RBAC given in Figure5.6

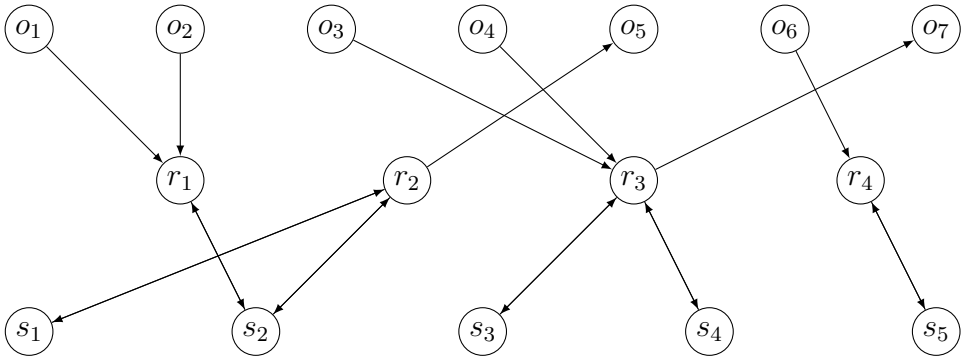


Figure 5.9. The output of the non-productive elimination strategy performed on RBAC given in Figure5.6

elimination. In this case, the total number of lost accesses is 2 objects. Note that, r_3 is not split because it can perform both $read$ and $write$ operations by itself.

5.3.3 Proactive Approach

As in DAC, the conservative strategies are static for RBAC. Although, this ensures complete security, it can potentially impact utility, since many permissions are revoked due to the potential of a leakage, even if in actuality the situation never occurs. In order to provide a less restrictive approach, we now propose a proactive strategy that follows the productive elimination described in Algorithm

5. Although it is technically possible to have a non-productive elimination, we do not propose it for the proactive approach because for each elimination, in contrast to conservative approach where $write$ permissions are revoked; $read$ permissions are revoked in proactive approach. As in the case of non-productive elimination some subjects lose access to objects unnecessarily. We assume that losing a $read$ access to an object is more crucial than losing a $write$ access and hence we do not propose a non-productive elimination.

The proactive approach revokes permissions only if the actions that could cause an unauthorized data flow actually happen. So, we continuously monitor read and write operations on the identified source and destination objects in $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ and $\mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$ that could potentially cause an unauthorized flow.

A confidentiality vulnerability $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$, is prevented by splitting the role r''' into two roles r''' and $r'''^{(C)}$ so that $o' \not\rightarrow_r r'''$, $o' \rightarrow_r r'''^{(C)}$, $\forall \hat{s} \in S \backslash \{s'\}$ such that $\hat{s} \leftrightarrow r'''$, assign $\hat{s} \leftrightarrow r'''^{(C)}$. Likewise an integrity vulnerability $(s', r', o, r'', s, r''', o') \in \mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$, is prevented by splitting the role r''' into two roles r''' and $r'''^{(C)}$ so that $r''' \not\rightarrow_w o'$, $r'''^{(C)} \rightarrow_w o'$, $\forall \hat{s} \in S \backslash \{s\}$ such that $\hat{s} \leftrightarrow r'''$, assign $\hat{s} \leftrightarrow r'''^{(C)}$.

We again observe the fact that altering SA and OA, might invalidate $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ or $\mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$ that has been computed via the identification algorithm given in Algorithm 4. Hence, $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ ($\mathcal{T}_{\mathcal{I}}^{\mathcal{R}}$, resp.) is recomputed if a $read$ ($write$, resp.) permission is used for the split.

The order of the $read$, $write$ actions that cause unauthorized data flow is also important. A elimination is only meaningful if a $read$ operation precedes a $write$

for confidentiality vulnerabilities and a $write$ operation precedes a $read$ operation for integrity vulnerabilities. Furthermore, we impose a *windowing* approach, stating that, if the succeeding event occurs after a predefined time window of size θ units, then the preceding event is considered to be outdated, so it is assumed that a data flow will not be unauthorized. Setting $\theta = \infty$ will eventually make the access configuration read-only for each destination object involved in a vulnerability path. .

The procedure for proactive approach for confidentiality vulnerabilities is given in Algorithm 7, which requires $O(|\mathcal{T}_C^{\mathcal{R}}||R|^2|S|)$ time for elimination and $O(|O|^2|R|^3|S|^2)$ time for recomputing $\mathcal{T}_C^{\mathcal{R}}$.

To show the behavior of the proactive approach, consider again the RBAC configuration shown in Tables 5.8 and 5.9, along with the potential sequence of operations shown in Table 5.10. Table 5.10 shows the proactive action for each operation in the sequence. Note that the proactive approach revokes a total of six permissions (2 each on operations (2), (3), and (5)).

Table 5.10. Sample Sequence of Actions and the behavior of Proactive Approach

	User's Operation	Proactive Action
1	s_2, r, o_1	
2	s_2, w, o_3	Split role r_3 into r_3 and $r_3^{(C)}$ so that $r_3 \rightarrow_w o_6$, $r_3 \rightarrow_w o_7$, $o_4 \rightarrow_r r_3$, $o_3 \rightarrow_r r_3^{(C)}$.
3	s_2, w, o_4	Split role r_3 into r_3 and $r_3^{(C')}$ so that $r_3 \rightarrow_w o_6$, $r_3 \rightarrow_w o_7$, $o_4 \rightarrow_r r_3$, $o_3 \rightarrow_r r_3^{(C')}$. Merge $r_3^{(C)}$ to $r_3^{(C')}$ since they share the same user assignment.
4	s_1, w, o_4	
5	s_3, r, o_3	Access denied

Algorithm 7 Algorithm for Proactive Strategy for RBAC

Input: $\mathcal{T}_C^{\mathcal{R}}$
Set the read file indicator $\mathcal{R}_o^s = 0, \forall s \in S, o \in O$.
Set δ as the current timestamp.
Monitor the access control actions done on the objects on vulnerability paths identified in $(o, r', s, r'', o', r''', s') \in \mathcal{T}_C^{\mathcal{R}}$ *- Use (Algorithm 4)*
if o is read by s **then**
    Set $\mathcal{R}_o^s = \delta$.
end if
if o' is written by s and $\delta - \mathcal{R}_o^s \leq \theta$ **then**
    if $\nexists r \in R$ such that $o' \rightarrow_r r$ is the only assigned permission **then**
        Create a new role, $r^* \in R$ and set $o' \rightarrow_r r^*$.
        for all $\hat{s} \in S \backslash \{s\}$ such that $\hat{s} \leftrightarrow r'''$ **do**
            Set $\hat{s} \leftrightarrow r^*$.
        end for
    end if
    for all $r_1, r_2 \in R$ **do**
        if $\nexists s$ such that ($s \leftrightarrow r_1$ and $s \not\leftrightarrow r_2$) or ($s \not\leftrightarrow r_1$ and $s \leftrightarrow r_2$) **then**
            Copy the permission assigned to r_2
            Delete role r_2
        end if
    end for
    Merge roles that have common subject assignments
    Redo identification algorithm (Algorithm 4) to recompute $\mathcal{T}_C^{\mathcal{R}}$.
end if

5.3.4 Retrospective Approach

A retrospective approach is also possible for RBAC. As discusses earlier in this chapter, the purpose of the retrospective approach is to eliminate (future) data leakages through auditing the access logs. This strategy is important especially when blocking access to certain files would cause significant consequences such as in a hospital. In this strategy, object read/write operations are logged and periodically, the log to date is analyzed to determine whether an unauthorized data flow has actually happened. If so, necessary permissions are revoked to block the data flow channels so that they will not be utilized for this purpose in the future.

In RBAC, due to the limitations of the ILP model, building an optimal strategy is not possible unless a similar approach discussed in Section 5.3.1 is utilized. Although, converting the RBAC into a DAC; utilizing the ILP model only with the constraints corresponding to the data flow channels; and finally using a Role Miner to obtain an RBAC decomposition would satisfy the requirements of the retrospective approach, performing these operations each time when the logs are audited can be costly to implement. Therefore, we utilize the heuristic approaches discussed in Section 5.3.2 instead.

The retrospective approach for RBAC requires the identification of potential vulnerabilities (Algorithm 4) as an initial step. Then, for every identified vulnerability $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$, if $\exists t_1 : o \rightarrow_r r' \rightarrow s$, $t_2 : s \rightarrow r'' \rightarrow_w o_d$, $t_3 : o' \rightarrow_r r''' \rightarrow s' \in \mathcal{L}$ where $t - \theta < t_1 < t_2 < t_3 < t$, spanning a time window of size θ, t_1, t_2, t_3 represent the timestamp of the logs and t represents the current timestamp, then $(o, r', s, r'', o', r''', s') \in \mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}}$, where $\mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}}$ is the set of

vulnerabilities to be removed. We provide this procedure in Algorithm 8, which requires $O(|\mathcal{L}||\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}|)$ time.

Algorithm 8 Algorithm for Retrospective Strategy

Input: $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$, and the log $\mathcal{L}$ related to tuples in $\mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$
Output: $\mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}}$.
Set the read file indicator $\mathcal{R}_o^s = 0$ and write file indicator $\mathcal{W}_o^s = 0, \forall s \in S, o \in O$ and $\mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}} = \emptyset$.
for all Logs $t : o \rightarrow_r s$ and $t : s \rightarrow_w o \in \mathcal{L}$ traced in sequence in the interval $(\delta - \theta, \delta)$ **do**
  if $(t, s, o, read)$ and $\exists(\hat{o}, r', \hat{s}, r'', \hat{o'}, r''', \hat{s'}) \in \mathcal{T}_{\mathcal{C}}^{\mathcal{R}}$ such that $(\hat{o} = o \wedge \hat{s} = s)$ or $(\hat{o'} = o \wedge \hat{s'} = s)$ **then**
    Set $\mathcal{R}_o^s = t$
    if $\hat{o'} = o$ and $\mathcal{R}_{\hat{o}}^{\hat{s}} < \mathcal{W}_{\hat{o'}}^{\hat{s}} < t$ **then**
      $\mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}} \leftarrow (\hat{o}, r', \hat{s}, r'', \hat{o'}, r''', \hat{s'})$
    end if
  end if
  if $t : s \rightarrow_w o$ and $\exists(\hat{o}, r', \hat{s}, r'', \hat{o'}, r''', \hat{s'}) \in \mathcal{T}_{\mathcal{C}}$ such that $\hat{o'} = o \wedge \hat{s} = s$ **then**
    Set $\mathcal{W}_o^s = t$
  end if
end for
Eliminate all vulnerabilities in $\mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}}$.

The elimination is handled using either a (1) productive or (2) non-productive in Algorithms 5 and 6, respectively, with the inputs replaced with $\mathcal{T}_{\mathcal{C}}^{\prime\mathcal{R}}$ and $\mathcal{T}_{\mathcal{I}}^{\prime\mathcal{R}}$ so that among all of the identified vulnerabilities, only the ones that have been utilized in a data leakage are eliminated.

Consider once again, the log of operations given in Table 5.10. In the retrospective approach, when auditing is done, based on the read and write actions, a data leakage is detected at item 5, where the actions $(s_2, r, o_1), (s_2, w, o_3), (s_3, r, o_3)$ is a confidentiality violation. The productive retrospective strategy will split the role r_2 into r_2 and $r_2^{(C)}$ so that $r_2 \not\rightarrow_w o_3$, $r_2^{(C)} \rightarrow_w o_3$, $s_1 \leftrightarrow r_2^{(C)}$. Hence, the num-

ber of roles increases by 1. On the other hand, the non-productive retrospective strategy will simply revoke $r_2 \rightarrow_w o_3$ causing s_1 to lose $write$ access to o_3.

5.3.5 Experimental Results for RBAC

We now present the experimental results to demonstrate the performance and restrictiveness of all of the proposed approaches for RBAC. We utilize four of the eight real life access control data sets with users and permissions – namely, (1) fire1, (2) fire2, (3) domino, (4) hc, since we were unable to run larger datasets due to hardware limitations. As before, these data sets encode a simple access control matrix denoting the ability of a subject to access an object (in any access mode). Thus, not only are these datasets for DAC, but also they do not have the information regarding which particular permission on the object is granted to the subject. So, we first perform a role mining operation on these datasets using the DEMiner [53] algorithm to obtain the SA and OA relations. Then, we assume that for all of the datasets that each assignment in OA represents both a read and a write permission on a distinct object. Due to the more complex nature of RBAC, we provide results for elimination of confidentiality vulnerabilities only, since the integrity vulnerabilities can be eliminated separately without affecting any confidentiality vulnerability (See Theorems 6 and Corollary 2).

The number of subjects and objects in the datasets along with the initial number of roles, vulnerabilities and number of assignments in SA and OA relations are given in Table 5.11.

For the optimal conservative approach (Section 5.3.1), we utilize the corresponding DAC solution methodology with the ILP model as per Figure 5.5.

Table 5.11. Dataset Details

Data Set	Size Sub.	Obj.	Roles	Vulnerabilities	Init $\|SA\|$	Init $\|OA\|$
1	365	709	86	185863	3843	1418
2	325	590	11	115792	1261	1180
3	79	231	38	18359	249	462
4	46	46	19	1117	433	92

Table 5.12. Results for Optimal Data Leakage Free Access RBAC

Data Set	CPLEX Time	DE Miner Time	Final $\|R\|$	% Change	Final $\|SA\|$	% Change	Final $\|OA\|$	% Change
1	67.866	0.5	91	0.06	3527	-0.08	1280	-0.10
2	0.075	0.256	13	0.18	1500	0.19	1024	-0.13
3	0.32	0.047	42	0.11	286	0.15	116	-0.75
4	0.24	0.02	21	0.11	461	0.06	92	0.00

The ILP model is again executed using IBM CPLEX (v 12.5.1) running through callable libraries within the code. Note that the ILP model is modified slightly to have constraints only for the confidentiality vulnerabilities (Constraints 1 and 2 in Figure 5.3). Then, DEMiner is utilized once again to decompose the DAC into RBAC. For the heuristic conservative approach, we use the identification algorithm (Algorithm 4) along with the described permission revocations. For the proactive approach, the identification and monitoring algorithm is implemented(Algorithm 7). For the retrospective approach, we use Algorithm 8. The algorithms are implemented in C and run on a Windows machine with 16 GB of RAM and Core i7 2.93 GHz processor.

Table 5.12 presents the experimental results for the Optimal Conservative approach. The column "CPLEX Time", shows the time required to run the ILP formulation given in Figure 5.5, whereas the column "DEMiner Time" reflects the time required to obtain an RBAC decomposition after running the ILP model.

Table 5.13. Results for Data Leakage Free RBAC with Productive Heuristic

Data Set	Iden. Time	Elm. Time	Final $\|R\|$	% Change	Number of Splits	Final $\|SA\|$	% Change	Final $\|OA\|$	% Change
1	7.23	46.35	166	48.19	395	17575	357.33	1103	-22.21
2	0.95	16.34	21	90.91	327	1391	10.31	863	-26.86
3	0.08	0.68	65	71.05	131	441	77.11	358	-22.51
4	0.01	0.50	38	100	46	1177	171.82	65	-29.35

Table 5.14. Results for Data Leakage Free RBAC with Non-Productive Heuristic

Data Set	Iden. Time	Elm. Time	Number of Lost Access	Final $\|SA\|$	% Change	Final $\|OA\|$	% Change
1	7.26	0.05	27826	3843	0.00	1023	-27.86
2	0.98	0.01	9288	1261	0.00	853	-27.71
3	0.09	0.004	346	249	0.00	331	-28.35
4	0.01	≈ 0	619	433	0.00	46	-50.00

The column "Final $|R|$" provides the updated number of roles as a result of the ILP model and the "% Change" column shows the percentage change in the number of roles. The next two columns depict the amount of change in SA after the ILP model and the last two columns depict the amount of change in OA after the ILP model. On average, there is a 7.83% increase in the number of assignments in SA and 24.46% decrease in the number of assignments in OA. Overall, there is a 8.32% decrease in the total number of assignments.

In Table 5.13, we provide experimental results for our heuristic approach with productive elimination for the conservative strategy. Here, the column "Iden. Time" shows the time to run Algorithm 4, on the other hand the column "Elm. Time" shows the time required to run Algorithm 5. The column "Number of Splits" reflects the number of times a role is split due to a possible vulnerability. When the results are examined, we see that there is an average increase of 154.14% in the number of assignments in SA, whereas there is an average de-

crease of 25.23% in the number of assignments in OA. Hence the overall average increase in the total number of assignments is 64.45%. Moreover there is an average of 77% increase in the number of roles as a result of productive elimination.

In Table 5.14, the results for the nonproductive elimination for the conservative strategy, so here the column "Elm. Time" shows the time required to run Algorithm 6. The column "Number of Lost Accesses" depict the number of subject-object pairs so that the subject loses access to a particular object unnecessarily due to a permission revoked from a role to eliminate a vulnerability caused by another subject. The results show that there is a decrease of 33.48% in the number of assignments in OA on the average. The number of roles and the number of assignments in SA remain unchanged as expected.

We run experiments for the proactive approach, as discussed in Section 5.3.3, and the eliminative actions occur on the fly. To test the relative performance of this approach, we have randomly generated a set of read/write operations that occur in the order they are generated. Similar to the case of DAC, we generate on average 100 operations for every subject (i.e., we generate $100 * |S|$ number of random operations). Then, we count the number of splits so far at the predefined intervals $10\% * |S|$, $50\% * |S|$, $100\% * |S|$, $1000\% * |S|$, $5000\% * |S|$, and finally $10000\% * |S|$. Table 5.15 gives the results. As we can see, the number of splits is steadily increasing but, they fall shy of that of the conservative method. However, we anticipate that as more object accesses occur, the number of splits in the proactive approach is likely to exceed that of the conservative approach. When we examine the results, we see that there is an average increase of 23.11% in the number of assignments in SA, whereas there is an average decrease of 24.82% in the

number of assignments in OA. In total, the overall average decrease in the number of assignments is 0.85%. Moreover there is an average of 37% increase in the number of roles as a result of productive elimination. Also, note that in the current set of experiments, we have set a window size of 1000 – this means that if the gap between a subject reading an object and then writing to another object is more than 1000 operations, then we do not consider a data flow to have occurred. This choice is clearly arbitrary and could be entirely removed, to ensure no data leakages.

Table 5.15. Results for Proactive approach

Data	Final	%	Final	%	Final	%	Number of Splits %					
Set	$\|R\|$	Chg.	$\|SA\|$	Chg.	$\|OA\|$	Chg.	10	50	100	500	1000	10000
1	150	42.67	6269	38.70	1202	-15.23	0	5	14	171	252	279
2	18	38.89	1635	29.66	880	-25.42	0	1	4	104	246	307
3	54	29.63	271	8.84	377	-18.40	0	1	1	15	73	101
4	31	38.71	499	15.24	55	-40.22	0	2	4	41	44	46

We evaluate the retrospective approach similar to the proactive approach, by using the same set of random operations generated for the proactive approach. The log after every 2000 read/write operations is audited and the sizes of SA and OA is computed along with the number of splits (for the productive retrospective approach). As expected, the retrospective approach makes fewer eliminative actions than the proactive approach. For example when we compare the productive retrospective approach to proactive approach, we see that the average number of splits is 79.25% for retrospective approach. This amount is 183% for the proactive approach. The average decrease in the number of assignments in OA is 15% for retrospective approach, whereas the same metric is 24.82% for the proactive approach, implying that the retrospective approach preserves more assignments, as expected. Finally, the non-productive retrospective approach revokes 17.12%

of the assignments in the OA. The results are on Tables 5.16 and 5.17.

Table 5.16. Results for Productive Retrospective Approach

Data Set	Final $\|R\|$	% Chg.	Final $\|SA\|$	% Chg.	Final $\|OA\|$	% Chg.	Number of Splits % 10	50	100	500	1000	10000
1	122	29.51	5492	30.03	1279	-9.80	0	0	0	49	153	166
2	19	42.11	2417	91.67	1113	-5.68	0	0	0	17	63	74
3	55	30.91	308	23.69	446	-3.46	0	0	0	0	13	25
4	33	42.43	710	63.97	52	-43.48	0	0	0	0	41	52

Table 5.17. Results for Non-Productive Retrospective Approach

Data Set	Number of Lost Access	Final $\|SA\|$	% Change	Final $\|OA\|$	% Change
1	0	3843	0.00	1267	-10.65
2	0	1261	0.00	1100	-6.78
3	2	249	0.00	437	-5.41
4	0	433	0.00	50	-45.65

5.4 Discussion

In this chapter, we presented our proposed approaches to solve the data leakage problem. Now, we outline our major contributions. We show that there is no need to compute the transitive closure of the data flows in a DAC or RBAC. In this chapter, we show that considering only the single step confidentiality and integrity vulnerabilities is sufficient enough to capture the all of the vulnerabilities in a given DAC or RBAC configuration. This important observation reduces the size of the ILP model and the number of objects monitored significantly.

Another interesting result is that a conservative approach might not always be the most restrictive. All other proposed approaches to handle the data leakage problem in the literature utilize a monitor to track users actions. Although it is intuitive to think that this approach is less restrictive than an approach that cuts all

possible potential data leakage channels, a long term object access monitoring and blocking on-the-fly might yield a more restricted system. We can see this observation clearly from the experimental results as the proactive approach revokes more permissions than the conservative approach after a certain number of read and write operations. Also a hybrid approach can be built to handle different sections of an access control system using a different strategies with different restrictiveness. For example a conservative approach is useful for system files, whereas a proactive approach could make more sense for user's personal files.

In retrospective approach, at first, can be considered as useless since it allows for the leakages to happen before restricting them for future use. However, it provides significant advantages especially under the situations when access to data is crucial. Note that although this approach is the least restrictive as a result of the experiments, a long run usage beyond shown in the experiments might invalidate this observation.

Our proposed methodology has the following limitations. Addition of new subjects/objects or addition of new permissions might invalidate the actions taken by the proposed approaches in this chapter. Hence, the ILP model, or the identification algorithms (Algorithm 2,4) should be run again to restore validity of the proposed approaches. Also, because of the nature of DAC being discretionary, all data leakage identification/prevention approaches including the ones proposed in the literature are prone to a "denial-of-service" type of an attack where a user intentionally creates an object and gives permissions to another user for the purpose of leaking data.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Summary of Contributions

In this dissertation, our major contribution is developing security analysis methodologies for the rights and data leakage problems. In particular, for the rights leakage problem, we propose an administrative model for Temporal RBAC and Spatio-temporal RBAC. In order to reduce the complexity of the model, we represent time as discrete and periodic intervals and locations as logical locations. We develop a flexible security analysis methodology based primarily on decomposing the TRBAC and STRBAC security analysis problems into smaller and more manageable RBAC security analysis problems. For the data leakage problem we investigate both confidentiality and integrity violating data leakages in DAC and RBAC models focusing on three different data leakage elimination strategies, namely, conservative, proactive and retrospective. We prove that the transitive closure is not necessary for the data leakage analysis. Instead, examining data flows among pairs of objects is sufficient for analysis purposes. This reduces the complexity of the problem significantly. We show that given an access control configuration obtaining a data leakage free access control matrix with minimal modification is an NP-Complete problem. We propose an Integer Linear Programming model for the conservative approach for DAC. We claim that the corresponding formulation

for RBAC is not likely to be linear, however it is decidable. We also propose more scalable heuristic alternatives for conservative app.

6.2 Future Work

Our proposed models for both rights leakage problem and data leakage problem point to many new problems to be tackled as a future work. These problems vary from minor additions to our existing models to standalone problems that will require new models.

Rule Schedule Strategy for STRBAC: Recall that our solution methodology for STRBAC utilizes the decomposition strategies that we propose for TRBAC. We use the role schedule strategy for the time and location based decomposition. The major assumption that we make to perform the role schedule strategy is the long run behavior of the system. More specifically, if the system is periodic in terms of time and one visits each location periodically, then the rule schedules of the rules can be omitted safely without loss of generality. However, we could not utilize the rule schedule approach for STRBAC because of the location dimension creating a more complex problem to handle. Not only is the time required for the constant regions taken into account, but also the time needed to travel among different locations should be known.

At this point we underline the difference between the time and the location dimensions. Although the time dimension is not at users' control, i.e., no one can change the time, the location dimension is indeed at users' control. This is important because an organized path of location changes could reduce the time required to fulfill the requirements of the *goal state* for the rights leakage problem,

making the goal state reachable faster. This observation is crucial to answer the short term safety questions. Therefore a more complex strategy that addresses this problem should be proposed to handle the rule schedule strategy for STRBAC.

Role Hierarchies for STRBAC: Role hierarchies for STRBAC model is also a part of our future work. The dynamic temporal role hierarchies can be defined for the spatial dimension so that the structure of the hierarchy and the permission inheritance properties will not only be determined by the temporal constraints but also with the spatial constraints.

Content Analysis in Data Leakage Identification and Elimination: The major drawback of our proposed data leakage methodologies is the fact that the actual data that is read or written is not taken into account. This means that even if a subject reads from an object and writes into another object, both identified as source and destination of a confidentiality violation path, a confidentiality violation will not happen if the data that is written to the destination object does not match with any part of the source object. Essentially, this means that our data leakage elimination strategies, including the least restrictive ones are, in fact, perform *redundant eliminations* due to the lack of content analysis. Although content analysis will reduce the efficiency of our proposed methodologies, it should be performed not to have any redundant eliminations and to preserve as many assigned permissions as possible.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD Conference*, 1993.

[2] S. Aich, S. Sural, and A. K. Majumdar. Starbac: spatiotemporal role based access control. In *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II*, OTM'07, pages 1567–1582, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] P. Ammann and R. Sandhu. Safety analysis for the extended schematic protection model. *Security and Privacy, IEEE Symposium on*, 0:87, 1991.

[4] V. Atluri and S. A. Chun. An authorization model for geospatial data. *IEEE Trans. Dependable Secur. Comput.*, 1(4):238–254, Oct. 2004.

[5] V. Atluri and A. Gal. An authorization model for temporal and derived data: securing information portals. *ACM Trans. Inf. Syst. Secur.*, 5(1):62–94, Feb. 2002.

[6] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat. Practical domain and type enforcement for unix. In *Security and Privacy,*

1995. Proceedings., 1995 IEEE Symposium on, pages 66–77. IEEE, 1995.

[7] E. Barka and R. Sandhu. Framework for role-based delegation models. In *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*, pages 168–176. IEEE, 2000.

[8] E. Barka, R. Sandhu, et al. A role-based delegation model and some extensions. In *Proceedings of the 23rd National Information Systems Security Conference*, volume 4, pages 49–58, 2000.

[9] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.

[10] E. Bertino, C. Bettini, and P. Samarati. A temporal authorization model. In *Proceedings of the 2nd ACM Conference on Computer and communications security*, CCS '94, pages 126–135, 1994.

[11] E. Bertino, P. Bonatti, and E. Ferrari. Trbac: A temporal role based access control model. *ACM Transactions on Information and System Security*, 4(3):191–233, 2001.

[12] W. Boebert, W. Young, R. Kaln, and S. Hansohn. Secure ada target: : Issues, system design, and verification. In *IEEE Symp. on Security and Privacy*.

[13] W. E. Boebert and R. Y. Kain. A further note on the confinement problem. In *Security Technology, 1996. 30th Annual 1996 International Carnahan Conference*, pages 198–202. IEEE, 1996.

[14] A. D. Brucker and H. Petritsch. Extending access control models with break-glass. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 197–206, New York, NY, USA, 2009. ACM.

[15] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for coalition-based access control (cbac). In *Proceedings of the seventh ACM symposium on Access control models and technologies*, SACMAT '02, pages 97–106, New York, NY, USA, 2002. ACM.

[16] J. Crampton. Cryptographic enforcement of role-based access control. In P. Degano, S. Etalle, and J. Guttman, editors, *Formal Aspects of Security and Trust*, volume 6561 of *Lecture Notes in Computer Science*, pages 191–205. Springer Berlin Heidelberg, 2011.

[17] J. Crampton and H. Khambhammettu. Delegation in role-based access control. *Computer Security–ESORICS 2006*, pages 174–191, 2006.

[18] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. *In proceedings of Symposium on Access Control Models and Technologies (SACMAT)*, pages 1–10, 2008.

[19] D. F. Ferraiolo, D. R. Kuhn, R. Chandramouli, and J. Barkley. Role-based access control (rbac). In *15th National Computer Security Conference*, pages 554–563, 1992.

[20] A. Ferrara, G. Fuchsbauer, and B. Warinschi. Cryptographically enforced rbac. In *Computer Security Foundations Symposium (CSF), 2013 IEEE 26th*, pages 115–129, June 2013.

[21] A. L. Ferrara, P. Madhusudan, T. L. Nguyen, and G. Parlato. Vac-verifier of administrative role-based access control policies. In *Computer Aided Verification*, pages 184–191. Springer, 2014.

[22] A. L. Ferrara, P. Madhusudan, and G. Parlato. Security analysis of access control policies through program verification. In *25th IEEE Computer Security Foundations Symposium*, 2012.

[23] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. drbac: Distributed role-based access control for dynamic coalition environments. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 411–432, Washington, DC, USA, 2002. IEEE Computer Society.

[24] M. Gasser. *Building a secure computer system*. Van Nostrand Reinhold New York, 1988.

[25] Q. Guo, J. Vaidya, and V. Atluri. The role hierarchy mining problem: Discovery of optimal role hierarchies. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, pages 237–246. IEEE, 2008.

[26] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, Aug. 1976.

[27] M. Jaume, V. V. T. Tong, and L. Mé. Flow based interpretation of access control: Detection of illegal information flows. In *Information Systems Security*, pages 72–86. Springer, 2011.

[28] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. *IEEE Trans. Dependable Secur. Comput.*, 5(4):242–255, Oct. 2008.

[29] A. K. Jones, R. J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, SFCS '76, pages 33–41, Washington, DC, USA, 1976. IEEE Computer Society.

[30] J. Joshi and E. Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 81–90, 2006.

[31] J. Joshi, E. Bertino, and A. Ghafoor. Hybrid role hierarchy for generalized temporal role based access control model. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, pages 951–956. IEEE, 2002.

[32] J. Joshi, E. Bertino, and A. Ghafoor. Temporal hierarchies and inheritance semantics for gtrbac. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 74–83. ACM, 2002.

[33] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role based access control model. *IEEE Transactions on Knowledge and Data*

Engineering, 17(1):4–23, 2005.

[34] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[35] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, SACMAT '04, pages 126–135, New York, NY, USA, 2004. ACM.

[36] N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security*, 9(4):391–420, 2006.

[37] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Policy decomposition for collaborative access control. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 103–112, New York, NY, USA, 2008. ACM.

[38] Z. Mao, N. Li, H. Chen, and X. Jiang. Trojan horse resistant discretionary access control. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 237–246. ACM, 2009.

[39] S. Marinovic, R. Craven, J. Ma, and N. Dulay. Rumpole: A flexible break-glass access control model. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, SACMAT '11, pages 73–82, New York, NY, USA, 2011. ACM.

[40] S. Mondal, S. Sural, and V. Atluri. Towards formal security analysis of GTRBAC using timed automata. In *ACM Symposium on Access Control*

Models and Technologies, pages 33–42, 2009.

[41] H. Petritsch. *Break-Glass: Handling Exceptional Situations in Access Control*. Springer Vieweg, 2014.

[42] R. Sandhu, V. Bhamidipati, and Q. Munawer. The arbac97 model for role-based admisitration of roles. *ACM Transactions on Information and System Security*, 2(1):105–135, 1999.

[43] R. S. Sandhu. The schematic protection model: its definition and analysis for acyclic attenuating schemes. *J. ACM*, 35(2):404–432, Apr. 1988.

[44] 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.

[45] A. Schaad, J. Moffett, and J. Jacob. The role-based access control system of a european bank: A case study and discussion. *In proceedings of ACM Symposisum on Access Control Models and Technologies*, pages 3 – 9, 2001.

[46] M. Soshi. Safety analysis of the dynamic-typed access matrix model. In *Computer Security - ESORICS 2000*, volume 1895 of *Lecture Notes in Computer Science*, pages 106–121. Springer Berlin / Heidelberg, 2000.

[47] S. D. Stoller, P. Yang, C. R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. *ACM*, pages 445–455, 2007.

[48] R. K. Thomas. Team-based access control (tmac): a primitive for applying role-based access controls in collaborative environments. In *Proceedings of*

the second ACM workshop on Role-based access control, RBAC '97, pages 13–19, New York, NY, USA, 1997. ACM.

[49] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd.

[50] E. Uzun, V. Atluri, S. Sural, J. Vaidya, G. Parlato, A. L. Ferrara, and M. Parthasarathy. Analyzing temporal role based access control models. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, SACMAT '12, pages 177–186, New York, NY, USA, 2012. ACM.

[51] E. Uzun, V. Atluri, J. Vaidya, and S. Sural. Analysis of trbac with dynamic temporal role hierarchies. In *Data and Applications Security and Privacy XXVII*, pages 297–304. Springer, 2013.

[52] E. Uzun, V. Atluri, J. Vaidya, S. Sural, A. L. Ferrara, G. Parlato, and P. Madhusudan. Security analysis for temporal role based access control. *Journal of Computer Security*, 22(6):961–996, 2014.

[53] E. Uzun, D. Lorenzi, V. Atluri, J. Vaidya, and S. Sural. Migrating from dac to rbac. In *Data and Applications Security and Privacy XXIX*, pages 69–84. Springer International Publishing, 2015.

[54] J. Wainer and A. Kumar. A fine-grained, controllable, user-to-user delegation method in rbac. In *Symposium on Access Control Models and Technologies: Proceedings of the tenth ACM symposium on Access control models and technologies*, volume 1, pages 59–66, 2005.

[55] J. Warner, V. Atluri, R. Mukkamala, and J. Vaidya. Using semantics for automatic enforcement of access control policies among dynamic coalitions. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 235–244, New York, NY, USA, 2007. ACM.

[56] M. Yannakakis. Node- and edge-deletion np-complete problems. In *STOC*, pages 253–264, 1978.

[57] M. Yannakakis. Edge-deletion problems. *SIAM J. Comput.*, 10(2):297–309, 1981.

[58] L. Zhang, G. Ahn, and B. Chu. A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):404–441, 2003.

[59] X. Zhang, S. Oh, and R. Sandhu. Pbdm: a flexible delegation model in rbac. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 149–157. ACM, 2003.

[60] J. Zimmermann, L. Mé, and C. Bidan. An improved reference flow control model for policy-based intrusion detection. In *Computer Security–ESORICS 2003*, pages 291–308. Springer, 2003.