Secrecy Resilience of Authorization Policies and Its Application to Role Mining

by

Qiang Guo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We propose and study a new property that we call *secrecy resilience* in the context of authorization policies that are used to secure information systems.

An authorization policy expresses whether a principal (e.g., a user or process) is allowed to exercise a privilege (e.g., read or write) on a resource (e.g., a device or file). Access control is a process by which authorizations are enforced. We address the problem that disclosure of portions of an authorization policy is a threat that needs to be mitigated and argue that the ease with which an adversary can learn such portions of a policy can be a property of the policy itself. We then introduce the term secrecy resilience as a quantitative measure of the computational hardness that such an adversary encounters. We instantiate secrecy resilience for authorization policy which could be expressed as access control policy and Role-Based Access Control (RBAC) policy, and more specifically, consider the problem of role mining, in which a policy expressed as an access matrix is converted to a RBAC policy. We present a number of analytical results while highlighting that underlying assumptions we make, with regards to a priori knowledge an adversary has, is an important consideration in any such analysis. We present also our results from an empirical study of role mining algorithms from the literature and two new "baseline" algorithms we propose. The results of our study suggest that when secrecy resilience is the objective, a role mining algorithm that performs well along a different criterion for goodness, e.g., minimization of roles (e.g., RBAC policy generated by User-Role Miner), does not necessarily perform well for some disclosure events. Moreover, under the assumptions we made for empirical study, for the disclosure event that the victim user has a role from the adversary, Permission-Role Miner is the best role mining algorithm from the standpoint of secrecy resilience.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Mahesh Tripunitara, for his continuing help and advice on my master thesis. I would also like to express my thanks to my readers, Professor Patrick Mitran and Professor John Thistle, for taking their time to review my work and provide with professional suggestions. Last but not least, I would like to express my sincere gratitude to my family for all their support and encouragement in the way of pursuing my master's degree.

Dedication

I would like to give my deepest gratitude and appreciation to my advisor, Professor Mahesh Tripunitara, for guiding me, and for helping me develop my research skills over the past few years.

Meanwhile, I will always be thankful to my wife, Meng Li for her unwavering love and support. Finally, I will give my whole love to my daughter, Zijia Guo. Her birth during the writing of the thesis brings me much joy in my life.

Table of Contents

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 Problem Statement . . . 2

1.2 Our Work . . . 2

1.3 Outline . . . 3

2 Background and Related Work 4

2.1 Access Control . . . 4

2.1.1 Principle of Access Control . . . 4

2.1.2 Access Control Subjects and Objects . . . 5

2.1.3 Access Control Process . . . 7

2.2 Role Based Access Control . . . 9

2.2.1 Design of RBAC . . . 10

2.2.2 Benefits of RBAC . . . 12

2.2.3 RBAC Research Overview . . . 13

3 Secrecy Resilience 14

3.1 Probability and Entropy . . . 14

3.1.1 Probability . . . 14

3.1.2 Entropy . . . 15
3.2 Secrecy Resilience . . . 17
3.2.1 Application to Access Matrix . . . 18
3.2.2 Application to Two Basic Role Mining Algorithms . . . 19

4 Application to Role Mining Algorithm **24**
4.1 Role Mining Algorithms . . . 24
4.1.1 Fast Miner(Fast) . . . 25
4.1.2 Dynamic Miner(DM) . . . 29
4.1.3 PairCount Miner(PC) . . . 31
4.2 Input Datasets . . . 32
4.2.1 Datasets from the Literature . . . 33
4.2.2 Generated Data . . . 33
4.3 Assumptions . . . 33
4.4 Experiment Analysis . . . 34
4.4.1 Experiment and Analysis for Event One . . . 35
4.4.2 Experiment and Analysis for Event Two . . . 38

5 Conclusion and Future Work **43**

References **44**

List of Figures

2.1 The access control process 7
2.2 RBAC96 module 10

List of Tables

4.1 Sizes of the real-world datasets presented 33
4.2 Worst-Case of Secrecy Resilience for Event One 35
4.3 Best-Case of Secrecy Resilience for Event One 36
4.4 Worst-Case of Secrecy Resilience for Event Two 39
4.5 Best-Case of Secrecy Resilience for Event Two 41

Chapter 1

Introduction

Computer systems contain large amounts of information, much of which is of a sensitive nature. It is necessary to be able to define what entities have access to this information and in what ways they can access it. These functions are variously known as authorization policy[13].

Organizations usually need to deal with authorization, which typically follows authentication from the standpoint of the security of a system. Authentication confirms a claimed identity, e.g., that Alice is indeed Alice. Authorization complements authentication — simply because both Alice and Bob are authenticated, legitimate users of the shared system does not necessarily mean that they both have access to every resource in the system. Authorization is used to specify who has access to what, and this is usually expressed in an authorization policy.

A simple and natural syntax for an authorization policy is an *Access Matrix*. An instance of an access matrix, $M[\cdot]$, comprises a set S of subjects, a set O of objects and a set of rights, R. The matrix M can then be seen as an encoding of a function $M\colon S \times O \to 2^R$, where 2^R is the power-set or set of all subsets of R.

Role Based Access Control (RBAC) is another syntax for an authorization policy. We can think of RBAC as also comprising a set of subjects or users U and a set of rights or permissions, P. However, rather than assigning each $u \in U$ directly to a $p \in P$, we introduce an indirection called a *role*.

Correspondingly, we have a set of roles, L. An instance of an RBAC policy, then, comprises two relations. One is the user-role relation $UA \subseteq U \times L$, and the other is the role-permission relation $PA \subseteq P \times L$. A user $u \in U$ is authorized to permission p if and only if there exists a role $r \in L$ such that $\langle u, r\rangle \in UA$ and $\langle p, r\rangle \in PA$. A RBAC policy is

typically visualized as a kind of graph, with nodes to represent each member of U, P and L, and edges to represent each member of UA and PA.

Employing RBAC is not only convenient but reduces the complexity of access control because the number of roles in an organization is significantly smaller than that of users. Moreover, the use of roles as authorization subjects, instead of users, avoids having to revoke and regrant authorizations whenever users change their positions and/or duties within the organization.

1.1 Problem Statement

It is often acknowledged that the greatest information security threat to an enterprise is not from outsiders, but from insiders, e.g., employees or contractors who are not fully trustworthy. Also, while authorization and access control are used to protect resources and assets, the authorization policy itself is often perceived as an asset. That is, enterprises may not want an outsider, or an insider who is not fully trusted, to know who has access to what. For example, if Alice is an insider and has some privileges, what is the probability with which she can infer some privileges of some other employee Bob? So we need to characterize a notion as some measure of resilience that an authorization policy has to discovery of components of it.

1.2 Our Work

In this study, we get the inspiration from the entropy in information theory and propose the notion of *secrecy resilience*. Informally, we characterize it as some measure of resilience that an authorization policy has to discovery of components of it.

Then with two baseline role mining algorithms we proposed (*User-Role Miner, Permission-Role Miner*) and three role mining algorithms from the literature [10] (*Fast Miner, Dynamic Miner and PairCount Miner*), we generate the corresponding RBAC policy from both generated datasets and datasets from literature. Finally we conduct experiment for the performance of secrecy resilience for the generated RBAC policy, and analysis the result with the best case and worst case of secrecy resilience.

1.3 Outline

The chapters of the thesis are organized as below.

In Chapter 2, we first introduce the notion of access control and RBAC policy. Then we overview the previous research about them.

In Chapter 3, we first introduce the probability models that we use in our experiment. Then we give a concise introduction to information entropy. After that, we propose the notion of secrecy resilience as a measure of resilience of an authorization policy. Finally, we introduce the application of secrecy resilience to access matrix and two basic role mining algorithms in this chapter.

In Chapter 4, except for the two basic role mining algorithms we built in previous chapter, first we set up three role mining algorithms (Fast Miner(Fast), Dynamic Miner(DM), PairCount Miner(Pair)) from literature to generate the corresponding RBAC policy. Then we give a concise description for the datasets we use in our experiment. Next we build some assumptions for the experiment. Finally, we analyse the result with the best-case and worst-case of secrecy resilience.

In Chapter 5, we first summarize the work in this thesis. Then we propose that the work could be continued from two aspects in order to supplement the research of secrecy resilience.

Chapter 2

Background and Related Work

In this chapter, we introduce access control policy and Role Based Access Control.

2.1 Access Control

Access control are the methods by which users gain access to resources to fulfill business needs. Business drivers should always be at the heart of any access control system, because even the most secure system is useless if it does not advance or support the goals of the organization [1].

2.1.1 Principle of Access Control

Access control is the formalization of those rules for allowing or denying access. Access control defines exactly who can interact with what, and what the subject may do during that interaction. It is based on the granting of rights, or privileges, to a subject with respect to an object.

When discussing access control, a "right" is a permission to perform a given action. In the preceding scenarios, the chief executive officer (CEO) of the company would have the right to interact directly with the executive, but the person who delivers the executive's mail would not. Your roommate or spouse would have the right to enter your house (as proven by ownership of a key to the door), while someone who lives down the street would not have permission to enter.

There are three principal components of any access control scenario:

- **Policies**: The rules that govern who gets access to which resources.
- **Subjects**: The user, network, process, or application requesting access to a resource.
- **Objects**: The resource to which the subject desires access

Any time you have to decide whether to allow or deny access by a subject to a resource, you have entered the access control problem domain.

A well-defined access control system consists of three elements:

- **Policies**: Rules developed by someone with a strong knowledge of the organization, its assets, goals, and challenges.
- **Procedures**: Nontechnical methods used to enforce policies.
- **Tools**: Technical methods used to enforce policies.

Organizations typically use procedures and tools together to enforce policies. For example, most companies have strict policies to determine who has access to personnel records. These records contain sensitive and confidential information that could be used to inflict serious harm on individual employees and the company as a whole if those records were compromised. The policy may state that only employees within the human resources department, with a specific need for the information contained within a given record, may have access to it.

To enforce this policy, the company has procedures that state that a record can only be given to employees with the proper credentials (the authentication process) who fill out a form stating their specific need for the information contained in the record they request. When the request is approved, the employees may be given a username and password to access the employee records intranet site (the authorization process). The intranet site, along with the username and password, are the tools required to grant access to personnel records.

2.1.2 Access Control Subjects and Objects

The subject in an access control policy is a person or another application requesting access to a resource such as the network, a file system, or a printer.

There are three types of subjects when it comes to access control for a specific resource:

- **Authorized**: Those who have presented credentials and have been approved for access to the resource.
- **Unauthorized**: Those who do not possess the proper credentials or do not have the appropriate privileges to access the resource.
- **Unknown**: Those who have not presented any credentials at all; it is unknown whether they should be given access or not.

The difference between an unknown person and an unauthorized one is timing. An unknown person is anonymous. They have not attempted to log in or access a restricted resource yet. As soon as an unknown person attempts to access a restricted resource, they must fall into one of the other two categories: authorized or unauthorized.

Four broad categories of technologies can be subjects for the purposes of access control:

- **Networks**: A network is a subject when a resource on one network requests access to a resource on another network. A firewall rule that authorizes access to the Internet might use the internal network as a subject, with the Internet as the object.
- **Systems**: A system is a subject when one system requests access to resources on another system or on the network. This usually happens when a PC attempts to access a printer across the network.
- **Processes**: A process is most commonly a subject when an application process requests low-level access to the file system.
- **Applications**: An application is a subject when it needs to access external resources such as a printer or the network.

A technology subject does not have a username and password the way a human subject might, but it does have the same authorized, unauthorized, or unknown status.

There are three main categories of objects to be protected by access controls:

- **Information**: Any type of data asset.
- **Technology**: Applications, systems, and networks.
- **Physical location**: Physical locations such as buildings and rooms.

Information is the most common asset in terms of IT access controls. You put passwords on databases and applications to ensure that only authorized users can access them. Technology objects are just as important, because a malicious user can easily compromise the integrity of data by attacking the technology that stores and uses it. If an unauthorized user gains access to a file server, that user can easily steal, delete, or change the data stored on the file server.

2.1.3 Access Control Process

There are three steps to the access control process:

1. **Identification**: The process by which a subject identifies itself to the access control.
2. **Authentication**: Verification of the subject's identity.
3. **Authorization**: The decision to allow or deny access to an object.

The second step usually happens behind the scenes, so the subject is really only aware of two stages: they enter their credentials and are either given or denied access to a resource. Figure 2.1 illustrates the access control process using human interaction as an example.

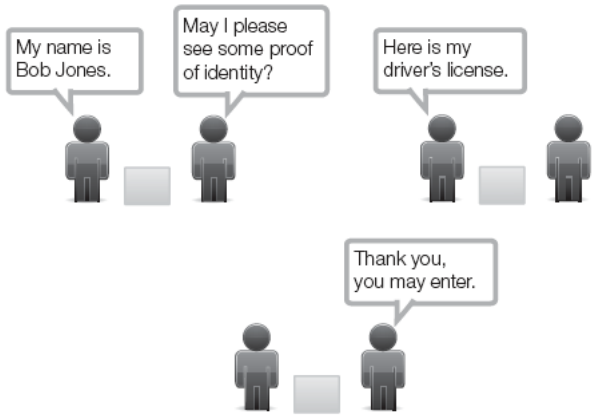

Figure 2.1: The access control process

2.1.3.1 Identification

The first step in any access control process is identification. The system must be able to apply labels to the two parts of the access equation: the subject and the object. In this case, a label is a purely logical description that is easy for the computer to understand. A human might easily recognize that "Beth" and "Elizabeth" are the same individual, but a computer cannot necessarily make that logical connection.

To make things simpler, you can assign a universal label to each subject and object. That label remains with that individual or resource throughout the life cycle of the privileged interaction with the object. The object also has a label to distinguish it from other resources. For example, a network might have six printers available, labeled "printer1","printer2" and so on. A person's label might be a user ID, his or her e-mail address, employee ID, or some other unique identifier.

The key is that each label must be unique, because it also provides accountability. When combined with the authentication system (which correlates the identified subject with the resources they are allowed to use) and system logging facilities, unique labels correlate subjects with their actions. This becomes especially important when trying to track down the cause of a system failure. This correlation relies on the trust between the subject and the access control policy. If you do not trust that a subject is who they say they are (and this trust is predicated on proof), the use of a uniquely identifying label is pointless.

2.1.3.2 Authentication

Authentication takes identification one step further by requiring proof of identity. There are many ways to authenticate a subject. The most common ones are:

- **Password**: A secret word or combination of characters that is known only to the subject. A good password is difficult to guess but easy for the subject to remember.
- **Token**: Something the subject has that no one else does, such as a smart card or a challenge-response device.
- **Shared secret**: Something only the subject and the authentication system know, such as the name of the subject's favorite pet or the mother's maiden name.

The key to both a password and a shared secret is secrecy. If the subject shares its password or shared secret information with someone else, the authentication system becomes less secure and the ability to correlate an action to a subject becomes less precise. Many

companies regulate this problem with a policy that an employee is personally responsible for anything done under his or her credentials. If an employee shares his credentials with a friend, for example, he is personally responsible for anything the friend might do.

Most authentication systems only require a single-stage authentication, but those protecting highly sensitive assets might use multiple factors. The three most common factors are:

- **Something you know**: Generally a password or shared secret.
- **Something you have**: A token or smart card ID badge.
- **Something you are**: Fingerprints or other biometric factors.

The last two factors are often used to provide or restrict physical access to secure buildings or rooms within buildings, although they can be used in access control systems protecting data as well.

Confidence in any authentication system can be measured by two components: the type of correlation and the number of authentication factors. A "retinal scan" (which is a biometric method) is inherently more secure than a simple password because it is much more difficult to copy or steal an eyeball than it is to guess or steal a password. Using more than one authentication factor increases the security of the system, because if one stage of the authentication system is compromised, the second can still restrict access to those who do not have the proper credentials.

2.2 Role Based Access Control

Privacy is increasing in importance since it becomes a major concern for both customers and enterprises in today's corporate marketing strategies. This raises challenging questions and problems regarding the use and protection of private messages.

One principle of protecting private information is based on who is allowed to access private information and for what purpose[12]. The workflow of any organization depends on the continuous and consistent execution of the assigned tasks by all the employees belonging to that organization.

The execution of these tasks, in turn, requires that each and every employee be given the necessary authorizations and privileges. Employees can acquire the relevant permissions based on some predefined rules, policies and mechanisms. These rules, policies and

mechanisms need to ensure not only that each user is given all the required permissions but also that no user is given any extra privilege. Failure to ensure the first aspect may lead to discontent among users or at most, may create some sort of hindrance in the smooth execution of tasks. However, failure to take care of the second aspect will most definitely lead to serious security breaches which can cause far more severe damages than displeasure or discontinuity in organizational workflow. Thus, the rules, policies and mechanisms need to be enforced properly so that none of the above mentioned adverse scenarios occur at any point of time.

2.2.1 Design of RBAC

Role-based access control(RBAC) is a policy that access control mechanism is defined around roles and privileges. The components of RBAC such as role-permissions, user-role and role-role relationships make it simple to perform user assignments. A study by NIST has demonstrated that RBAC addresses many needs of commercial and government organizations[8]. RBAC can be used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions. Although RBAC is different from MAC and DAC access control frameworks, it can enforce these policies without any complication. Figure shows a RBAC96 module [20].

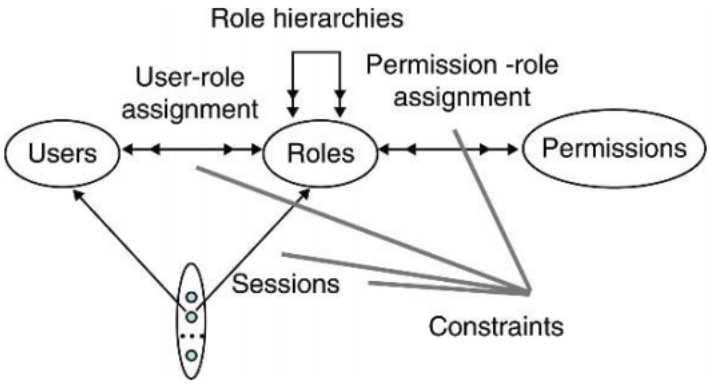

Figure 2.2: RBAC96 module

Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members or staff (or other system users) are assigned particular roles, and through those role assignments acquire the permissions needed to perform particular system functions. Since users are not assigned

permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user's account; this simplifies common operations, such as adding a user, or changing a user's department.

RBAC interference is a relatively new issue in security applications, where multiple user accounts with dynamic access levels may lead to encryption key instability, allowing an outside user to exploit the weakness for unauthorized access. Key sharing applications within dynamic virtualized environments have shown some success in addressing this problem[11].

Three primary rules are defined for RBAC:

1. Role assignment: A subject can exercise a permission only if the subject has been assigned with a role.

2. Role authorization: A subject's role must be authorized for the subject. This rule ensures that users can take on only roles for which they are authorized.

3. Permission authorization: A subject can exercise a permission only if the permission is authorized for the subject's active role. With rules 1 and 2, this rule ensures that users can exercise only permissions for which they are authorized.

Additional constraints may be applied as well, and roles can be combined in a hierarchy where higher-level roles subsume permissions owned by sub-roles.

When defining an RBAC model, the following conventions are useful:

- S = Subject = A person or automated agent.
- R = Role = Job function or title which defines an authority level.
- P = Permissions = An approval of a mode of access to a resource.
- SA = Subject Assignment.
- PA = Permission Assignment.
- RH = Partially ordered Role Hierarchy. RH can also be written: $\geq$ (The notation: $x \geq y$ means that x inherits the permissions of y.)
  - A subject can have multiple roles.
  - A role can have multiple subjects.
  - A role can have many permissions.

- A permission can be assigned to many roles.
- An operation can be assigned to many permissions.
- A permission can be assigned to many operations.

The RBAC policy could be described with set theory notation:

- $PA \subseteq P \times R$ and is a many to many permission to role assignment relation.
- $UA \subseteq U \times R$ and is a many to many user to role assignment relation.
- $RH \subseteq R \times R$

A subject may have multiple simultaneous sessions with/in different roles.

2.2.2 Benefits of RBAC

There are a number of benefits to using RBAC to restrict unnecessary authorization access based on people's roles within an organization, including:

- Improving operational efficiency. With RBAC, companies can decrease the need for paperwork and password changes when they hire new employees or switch the roles of existing employees. RBAC lets organizations quickly add and change roles, as well as implement them across platforms, operating systems (OSes) and applications. It also cuts down on the potential for error when user permissions are being assigned. Additionally, with RBAC, companies can more easily integrate third-party users into their networks by giving them predefined roles.
- Enhancing compliance. Every organization must comply with local, state and federal regulations. Companies generally prefer to implement RBAC systems to meet the regulatory and statutory requirements for confidentiality and privacy because executives and IT departments can more effectively manage how the data is accessed and used. This is particularly important for financial institutions and healthcare companies that manage sensitive data.
- Giving administrators increased visibility. RBAC gives administrators and managers more visibility and oversight into the business, while also guaranteeing that authorized users and guests on the system are only given access to what they need to do their jobs.

- Reducing costs. By not allowing user access to certain processes and applications, companies may conserve or more cost-effectively use resources, such as network bandwidth, memory and storage.
- Decreasing risk of breaches and data leakage. Implementing RBAC means restricting access to sensitive information, thus reducing the potential for data breaches or data leakage.

2.2.3 RBAC Research Overview

The roles in the RBAC strategy accurately reflect the activities, responsibilities, and functions of an organization. Therefore, many role engineering methods, role mining algorithms, hierarchical role construction algorithms, and least privilege assignment algorithms are born to generate role sets.

The reason that the RBAC strategy is safe is to avoid conflicts of interest with the separation of duties. Static Separation of Statistic Separation of Duty (SSD) is the introduction of constraints in the user-role assignment process and the role-role inheritance relationship. Dynamic Separation of Duty (DSD, Dynamic Separation of Duty) introduces authority constraints during user session activation, and the method of conflict detection of separation of duties is also very popular among researchers. With the continuous expansion of the scope of RBAC applications, new problems will continue to arise, such as the RBAC strategy based on time constraints. In 2005, the GTRBAC model proposed by Joshi et al[5] gave the simple method of SoD based on time constraints. By definition, there are two common types: periodic time constraints and continuous time constraints[6]. Each type of time constraint includes weak constraints, strong constraints, and super-strong constraints. Yamazaki et al. enumerated in more detail various SoD constrained by overtime[19, 2]. Chadwick et al. proposed a multi-session SoD (MSoD) constraint. In some virtual institutions, users need to access resources in multiple domains, and the user-role assignment relationship is also done by administrators of multiple domains. Therefore, no single RBAC system can know all the roles a user has.

Chapter 3

Secrecy Resilience

In this chapter, we first introduce the probability that we use in our experiment. Then we give a concise description for information entropy. Next we propose a measure of secrecy resilience to characterize an authorization policy. Finally, we introduce the application of secrecy resilience to access matrices and two basic role mining algorithms in this chapter.

3.1 Probability and Entropy

3.1.1 Probability

Probability is the branch of mathematics concerning numerical descriptions of how likely an event is to occur, or how likely it is that a proposition is true. The probability of an event is a number between 0 and 1, where, roughly speaking, 0 indicates impossibility of the event and 1 indicates certainty. The higher the probability of an event, the more likely it is that the event will occur.

In the experiment of this thesis, we assume that the event that victim b is assigned to at least one role in $\{r_1, \ldots, r_k\}$ follows the *inclusion–exclusion principle*. That is, the probability that *victim* b is assigned to at least one role in $\{r_1, \ldots, r_k\}$ is:

$$P[\bigcup_{i=1}^{n} A_i] = \sum_{i=1}^{n} P[A_i] - \sum_{1 \le i < j \le n} P[A_i \cap A_j] + \sum_{1 \le i < j < k \le n} P[A_i \cap A_j \cap A_k] - \cdots + (-1)^{n-1} P[\bigcap_{i=1}^{n} A_i]$$

A_i is the event that *victim* b has role r_i.

Meanwhile, we also assume that permissions are orthogonal to one another, i.e., possession of permission p has no implications for possession of another permission q, and possession of a permission on a resource r has no implications for possession of a permission on another resource s. Hence we can assume that the adversary user makes uniformity and independence assumptions on user-permission assignment in the access matrix, i.e., $p = \Pr\{u \text{ is assigned permission } p\} = 1/2$ and the event that u has permission p_i and u has permission p_j are independent for every $i \neq j$. So P[u is assigned with permissions $p_{i_1}, \ldots, p_{i_2}$] $= 1/2^N$, where N is the number of permissions.

3.1.2 Entropy

Entropy in information theory is directly analogous to the entropy in statistical thermodynamics. The analogy results when the values of the random variable designate energies of microstates. Entropy has relevance to other areas of mathematics such as combinatorics. The definition can be derived from a set of axioms establishing that entropy should be a measure of how "surprising" the average outcome of a variable is. For a continuous random variable, differential entropy is analogous to entropy.

The development of the idea of entropy of random variables and processes by Claude Shannon provided the beginnings of information theory. We shall see that entropy and related information measures provide useful descriptions of the long term behavior of random processes and that this behavior is a key factor in developing the coding theorems of information theory.

3.1.2.1 Introduction of Information Entropy

The basic idea of information theory is that the "informational value" of a communicated message depends on the degree to which the content of the message is surprising. If an event is very probable, it is of no surprise (and generally uninteresting) when that event happens; hence transmission of such a message carries very little new information. However, if an event is unlikely to occur, it is much more informative to learn that the event happened or will happen. For instance, the knowledge that some particular number will not be the winning number of a lottery provides very little information, because any particular chosen number will almost certainly not win. However, knowledge that a particular number will win a lottery has high value because it communicates the occurrence of a very low probability event.

The information content of an event E is a function which decreases as the probability $p(E)$ of an event increases, defined by $I(E) = -log_2(p(E))$ or equivalently $I(E) =$

$\log_2(1/p(E))$, where *log* is the logarithm. Entropy measures the expected (i.e., average) amount of information conveyed by identifying the outcome of a random trial. This implies that casting a die has higher entropy than tossing a coin because each outcome of a die toss has smaller probability (about $p = 1/6$ than each outcome of a coin toss ($p = 1/2$)).

Shannon's theorem also implies that no lossless compression scheme can shorten all messages. If some messages come out shorter, at least one must come out as long as the original message due to the pigeonhole principle. In practical use, this is generally not a problem, because one is usually only interested in compressing certain types of messages, such as a document in English, as opposed to gibberish text, or digital photographs rather than noise, and it is unimportant if a compression algorithm makes some unlikely or uninteresting sequences larger.

3.1.2.2 Definition of Information Entropy

Named after Boltzmann's H-theorem, Shannon defined the entropy H of a discrete random variable X with possible values $\{x_1, ..., x_n\}$ and probability mass function $P(X_i) = P[X = X_i]$ as [16]:

$$H(X) = E[I(X)] = E[-log(P(X))]$$

Here E is the expected value operator, and I is the information content of X. $I(X)$ is itself a random variable.

The entropy can explicitly be written as [16]:

$$H(X) = -\sum_{i-1}^{n} P(x_i) log_b P(x_i)$$

where b is the base of the logarithm used. Common values of b are 2, Euler's number e, and 10, and the corresponding units of entropy are bits for $b = 2$, nats for $b = e$, and digits for $b = 10$.

In the case of $P(x_i) = 0$ for some i, the value of the corresponding summand $0 * log_b(0)$ is taken to be 0, which is consistent with the limit:

$$\lim_{p \to 0^+} p * log(p) = 0$$

3.1.2.3 Characterization of Information Entropy

To understand the meaning of $-\sum p_i log(p_i)$, first define an information function I in terms of an event i with probability p_i. The amount of information follows from Shannon's solution to the fundamental properties that information should have:

1. $I(p)$ is monotonically decreasing in p: an increase in the probability of an event decreases the information from an observed event, and vice versa.

2. $I(p) \geq 0$: information is a non-negative quantity.

3. $I(1) = 0$: events that always occur do not communicate information.

4. $I(p_1 p_2) = I(p_1) + I(p_2)$: the information learned from two independent events is the sum of the information learned from each event.

3.2 Secrecy Resilience

It is often recognized that the biggest information security threat to an organization does not come from outsiders, but from insiders, such as employees or contractors who are not entirely trustworthy. In addition, although authorization and access control are used to protect resources and assets, the authorization policy itself is usually regarded as an asset. In other words, the organization may not want outsiders or insiders if does not fully trust to know any piece of the user-permission mapping picture.

Inspired by the notion and implementation of information entropy, we propose the notion of secrecy resilience. To define secrecy resilience, we first adopt an event that is a disclosure of a part of the authorization policy that is of interest to an adversary. We call this the *disclosure event*.

Two examples of a disclosure event are:

- Given an RBAC user a who is the adversary and a role r, the event that a victim user is a member of the role r.

- Given an RBAC user a who is the adversary, the event that a victim user is a member of some roles of which a is a member.

Once we identify a disclosure event, there are only two possibilities for it: either the event occurs, or it does not, i.e., it is natural to associate a random variable, denote it X, which takes on the value true if the disclosure event occurs and false if it does not. Denote as $\Pr\{X = \mathsf{true}\}$ the probability of the former and $\Pr\{X = \mathsf{false}\}$ the probability of the latter.

Now, we define secrecy resilience as follows:

Definition 3.2.1 (Secrecy Resilience)**.** Given (i) an authorization policy $\mathcal{P}$, (ii) a disclosure event t, suppose $X \in \{\mathsf{true}, \mathsf{false}\}$ is a random variable that takes on the value true if the disclosure event t occurs, and false otherwise. Then, we say that the secrecy resilience of $\langle \mathcal{P}, t \rangle$ is:

$$S(X) = -\Pr\{X = \mathsf{true}\} \cdot \log_2(\Pr\{X = \mathsf{true}\}) - \Pr\{X = \mathsf{false}\} \cdot \log_2(\Pr\{X = \mathsf{false}\})$$

From the above definition for secrecy resilience, the two events: 1) the disclosure event it true and 2) the disclosure event is false are mutually exclusivity. As a result, we have two theorems:

Theorem 1. When it is ascertained that that the disclosure event t is true, i.e,. $\Pr\{X = \mathsf{true}\} = 1$ and $\Pr\{X = \mathsf{false}\} = 0$, or the disclosure event t is false, i.e,. $\Pr\{X = \mathsf{true}\} = 0$ and $\Pr\{X = \mathsf{false}\} = 1$, the secrecy resilience of $\langle \mathcal{P}, t \rangle = 0$

Theorem 2. When there is no information to decide whether the disclosure event t is true or false, i.e,. $\Pr\{X = \mathsf{true}\} = 1/2$ and $\Pr\{X = \mathsf{false}\} = 1/2$, the secrecy resilience of $\langle \mathcal{P}, t \rangle = 1$

From the above theorems, we can see that theorem 1 and theorem 2 set the boundary of secrecy resilience. So we have another theorem:

Theorem 3. For any disclosure event t in an authorization policy, the secrecy resilience of $\langle \mathcal{P}, t \rangle$ is always greater than or equal to 0 and less than or equal to 1.

The value of $\Pr\{X = \mathsf{true}\}$ and $\Pr\{X = \mathsf{false}\}$ is between 0 and 1 and $\Pr\{X = \mathsf{true}\} + \Pr\{X = \mathsf{false}\} = 1$, so that the value of $S(X)$ could be no less than 0 and no more than 1.

Since for any disclosure event in an authorization policy, if the secrecy resilience is close to 1, then we could say that the policy performs well in secrecy resilience. Or if the secrecy resilience is close to 0, then we could say that the policy performs badly in the secrecy resilience. So when we choose or set up the authorization policy, the secrecy resilience can be a measure to justify the risk form the inside. What's more, the organization can use it to improve or comply with the inside risk.

3.2.1 Application to Access Matrix

In access matrix, the disclosure event could be: given an access matrix user a is the adversary, the event that a victim user opposes a member of a permission of which a has. Then we could have a theorem for the application to access matrix.

Here are two assumptions we make for the theorem.

- Given an access control matrix D.
- Permissions are orthogonal to one another, i.e., possession of permission p has no implications for the possession of another permission q, and possession of a permission on a resource r has no implications for the possession of a permission on another resource s. Hence adversary user has uniformity and independence assumptions in the access matrix, i.e., $p = \Pr\{u \text{ is assigned with permission } p\} = 1/2$ and the event that u has permission p_i and the u has permission p_j are independent for every $i \neq j$.

Theorem 4. With the assumptions we adopted, the secrecy resilience of the disclosure event t in an access matrix is always 1.

For the disclosure event t in an access matrix, since we make uniformity and independence assumptions, the $\Pr\{u \text{ is assigned with permission } p\} = 1/2$, i.e., $\Pr\{X = \mathsf{true}\} = \Pr\{X = \mathsf{false}\} = 1/2$, so that the secrecy resilience of the disclosure event in an access matrix is always 1. That also means access matrix performs well in secrecy resilience.

3.2.2 Application to Two Basic Role Mining Algorithms

First we set up and introduce two basic role mining algorithms: User-Role Miner and Permission-Role Miner.

3.2.2.1 User-Role Miner (UR)

The User-Role Miner consists of two phases, described as below:

1. **Generation of roles**: Firstly, we identify the permission set for each user, then we create a new role for each distinct permission set. The process could also be described as follows: we suppose $P = \{p_1, p_2, \ldots, p_n\}$ is the set of all permissions, let $S \subseteq P$ be some subset of P. If some user u's set of permissions is exactly the set S in the access matrix, then we create a role r_S.
2. **Generation UA and PA**: We assign all the permissions in S to the newly created r_S. Then, for every user u whose permission set is S, we assign that user to role r_S.

Algorithm 1 User-Role Miner Algorithm

Input: Access Matrix $D \equiv (U, P, UP)$
Output: $\langle R, UA, PA \rangle$

```
1  GenRoles ← ∅, UA ← ∅, PA ← ∅, R ← ∅, PermSet ← ∅
2  {Generation of roles}
3  foreach user u ∈ U do
4      if R(P(u)) ∉ GenRoles then
5          GenRoles ← GenRoles ∪ {R(P(u))}
6  {Generation of UA and PA}
7  foreach user u ∈ U do
8      foreach Role(x) ∈ GenRoles do
9          R ← R ∪ Role(x)
10         if x ≡ P(u) then
11             UA ← UA ∪ {⟨u, Role(x)⟩}
12 foreach Role(x) ∈ GenRoles do
13     foreach permission p ∈ P do
14         if p ∈ x then
15             PermSet ← PermSet ∪ p
16         PA ← PA ∪ {⟨Role(x), PermSet⟩}
17 return ⟨GenRoles, UA, PA⟩
```

Algorithm 1 gives the detailed steps. The first phase consists of lines 3-5. The for loop in line 3 iterates over all users while if the permission set of user u is not in the generated role set $GenRoles$, then add the permission set as a new role $R(P(u))$ in the generated role set $GenRoles$.

The second phase consists of lines 7-17. The for loop in line 7 iterates over all users in the U set. Then the for loop in line 8 iterates over all generated roles $Role(x)$ in the $GenRoles$ set. If the permission set x of the $Role(x)$ is just the same as the permission set of user u, then add the user-role tuple $\langle u, Role(x) \rangle$ to the user-role set UA. After that, the for loop in line 12 iterates over all generated roles $Role(x)$ in the $GenRoles$ set. Then the for loop in line 13 iterates over all permissions p in the permission set P. If the permission p is in the permission set of $Role(x)$, then add permission p to the permission set of $PermSet$. Finally, after the iteration in line 13, add the role-permission tuple $\langle Role(x), PermSet \rangle$ to the role-permission set PA.

3.2.2.2 Permission-Role Miner (PR)

The Permission-Role Miner consists of two phases, described below:

1. **Generation of PA**: We first identify each of the permissions in the permission set. That is, suppose $P = \{p_1, p_2, \ldots, p_n\}$ is the set of all permissions. Then, for every permission $p_i \in P$, we assign a role r_i to it.

2. **Generation of UA**: For each user u in the U set, if the permission set of the user u is $P(u)$, then for each $p_i \in P(u)$, we assign role r_i to user u.

Algorithm 2 Permission-Role Miner Algorithm

Input: Access Matrix $D \equiv (U, P, UP)$
Output: $\langle R, UA, PA \rangle$

```
1  R ← ∅, UA ← ∅, PA ← ∅
2  {Generation of PA}
3  foreach permission p_i ∈ P do
4      R ← R ∪ {r_i}
5      PA ← PA ∪ {⟨p_i, r_i⟩}
6  {Generation of UA}
7  foreach user u ∈ U do
8      RoleSet ← ∅
9      foreach p_i ∈ P(u) do
10         RoleSet ← RoleSet ∪ r_i
11     UA ← UA ∪ {⟨u, RoleSet⟩}
12 return⟨R, UA, PA⟩
```

Algorithm 2 gives the detailed steps. The first phase consists of lines 2-5. The for loop in line 3 iterates over all permission p_i in the permission set P. For each permission p_i in the permission set P, generate and assign a role r_i with it. Then add the role r_i to the role set R. Meanwhile, add the tuple $\langle p_i, r_i \rangle$ to the permission-role set PA.

The second phase consists of lines 7-11. The for loop in line 7 iterates over all users u in the U set. For each user u, the permission set of it is $P(u)$. Then for each permission p_i in the permission set $P(u)$, assign the corresponding r_i to the *RoleSet*. Finally, after the loop in line 9, add the tuple $\langle u, RoleSet \rangle$ to the user-role set UA.

3.2.2.3 Assumption and Theorem

Here are the assumptions for the theorem of User-Role Miner:

- Given an access control matrix D.
- Permissions are orthogonal to one another, i.e., possession of permission p has no implications for the possession of another permission q, and possession of a permission on a resource r has no implications for the possession of a permission on another resource s. Hence adversary user has uniformity and independence assumptions in the access matrix, i.e., $p = \Pr\{u \text{ is assigned with permission } p\} = 1/2$ and the event that u has permission p_i and the u has permission p_j are independent for every $i \neq j$.
- For users and permissions in D, the User-Role Miner algorithm is used to generate roles and the RBAC policy.
- The disclosure event t is: given a RBAC policy user a who is the adversary, the event that a victim user is a member of some roles of which a is a member.

Theorem 5. Suppose k is the number of permissions the adversary has. With the assumptions we adopted, the secrecy resilience of the disclosure event t in RBAC policy which is generated by User-Role Miner is:

$$S(X) = -(1/2)^k \cdot \log_2 (1/2)^k - (1 - (1/2)^k) \cdot \log_2 (1 - (1/2)^k)$$

That's because for the RBAC policy generated by User-Role Miner, adversary user could only be assigned with one role, which means this role has all of the permissions that user has. So the probability that the victim has this role is $(1/2)^k$, where k is the number of permissions the adversary user has.

Here are the assumptions for the theorem of Permission-Role Miner:

- Given an access control matrix D.
- Permissions are orthogonal to one another, i.e., possession of permission p has no implications for the possession of another permission q, and possession of a permission on a resource r has no implications for the possession of a permission on another resource s. Hence adversary user has uniformity and independence assumptions in the access matrix, i.e., $p = \Pr\{u \text{ is assigned with permission } p\} = 1/2$ and the event that u has permission p_i and the u has permission p_j are independent for every $i \neq j$.

- For users and permissions in D, using Permission-Role Miner algorithm to generate roles and the RBAC policy.
- The disclosure event t is given a RBAC policy user a who is the adversary, the event that a victim user is a member of some roles of which a is a member.

Theorem 6. Suppose k is the number of roles the adversary has. With the assumptions we adopted, the secrecy resilience of the disclosure event t in RBAC policy which is generated by Permission-Role Miner is:

$$S(X) = -(1/2)^k \cdot \log_2 (1/2)^k - (1 - (1/2)^k) \cdot \log_2 (1 - (1/2)^k)$$

That's because for the RBAC policy generated by Permission-Role Miner, each permission will be assigned with one role. So if the adversary user has k permissions, then it will be assigned with k roles in the RBAC policy. So the probability that the victim doesn't have a role which the adversary has is $(1/2)^k$, where k is the number of roles the adversary user has.

Chapter 4

Application to Role Mining Algorithm

In this chapter, we first build three role mining algorithms from the literature (Fast Miner(Fast), Dynamic Miner(DM) and PairCount Miner(Pair)) and generate the flat RBAC policy from an access matrix. Then, both the generated datasets and the datasets from previous role mining literature are introduced. Finally, we discuss the case study of secrecy resilience and the application to these role mining algorithms.

4.1 Role Mining Algorithms

Recently, there has been increasing interest in role mining, which uses data mining techniques to discover roles from existing system configuration data. Because role mining uses automated techniques, it has the potential to accelerate the role engineering process, which is the costliest part of migrating to a RBAC system. While many role mining algorithms have been proposed in recent years.

Most of the existing role mining algorithms can be divided into two classes based on their output. The first class outs a prioritized list of candidate roles, each of which is a set of permissions. These algorithms output a vector of candidate roles C ordered by their priority. Examples include Complete Miner and Fast Miner in [18]. While the algorithms in the second class output a complete RBAC state. Examples include ORCA Miner [14], Graph Optimization Miner [21], Role Edge Minimization Miner [3], and Hierarchical Miner [9].

In the experiment of this thesis, except for the User-Role Miner and Permission-Role Miner which we introduce in the previous chapter, we also set up Fast Miner(Fast), Dynamic Miner(DM) and PairCount Miner(Pair) to generate the flat RBAC policy.

4.1.1 Fast Miner(Fast)

Fast Miner(Fast) was proposed by Vaidya et al.[18] in 2006. It starts by creating an initial set of roles from the distinct user permission sets. It then computes all possible intersection sets of all pairs of roles in the initial roles set. Specifically, the set of candidate roles generated by Fast Miner is: $\bigcup\{\bigcap_{pair\ of\ initial\ Roles} R_{initial}\}$.

Vaidya et al.[18] proposed a role prioritization method of a candidate role r as:

$$|e(r)| \times \alpha + |n(r)|$$

where *e(r)* denotes the set of users that have exactly the permissions in r, *n(r)* is the number of users whose permissions are a superset of r, and α is a tunable parameter to favor initial roles.

The time complexity of Fast Miner is $O(n^2m)$.

The Fast Miner consists of two phases, described below:

1. **Generation of Roles**

   (a) **Identification of Initial Set of Roles**: In this phase, we group all users who have the exact same set of permissions. This can be done in a single pass over the data by maintaining a hash table of the sets of permissions seen. This significantly reduces the size of the data set since all users who have the same roles have the exact same permissions. These form the initial set of roles, say *InitRoles*.

   (b) **Subset Enumeration**: In this phase, we determine all of the potentially interesting roles by computing all possible intersection sets of all pairs of roles created in the initial phase. Let this set be *GenRoles*. Each unique intersection set is added to the set of generated roles (thus only the unique set of intersections is maintained). Though the number of intersections is equal to the size of the power set of the initial roles ($C^2_{InitRoles}$), the actual roles enumerated are much smaller (since many intersections result in the empty set, or in the same intersection set). Note that the generated roles are the maximal set of interesting roles.

(c) **User Count Computation**: In this phase, for each generated role in *GenRoles*, we count the number of users who have the permissions associated with that role. We actually maintain two sets of counts: (i) the *orig_count(i)*, the original number of users who have exactly the set of permissions corresponding to role i and nothing else, and (ii) *count(i)*, an updated count of users whose permissions are a superset of the permissions associated with this role i. It should be obvious that each initial role will have an updated count greater than its original count if and only if it is a subset of one of the other initial roles.

2. **Role Prioritization and Generation of UA, PA**: In the algorithm, we could typically identify many potential role, these need to be prioritized/ordered in some way. One possibility is to simply order the roles according to the number of users having that role. That is, we use the following predicate to sort: $(orig\_count * priority + Count)$. Here, priority is simply the multiplication factor used to bias the results towards the roles found in the initial phase (i.e., do not report a generated set as a role unless it is really interesting). Thus, this ensures that a well supported original role does not get lost in the chaff of generated roles (i.e., roles that are not part of $InitRoles$). We experimented with 1 as the value for priority.

Algorithm 3 Fast Miner Algorithm - Phase One

Input: Access Matrix $D \equiv (U, P, UP)$
Output: $GenRoles, InitRoles, Count, orig\_count$

```
1  InitRoles ← ∅, GenRoles ← ∅
2  {Identification of Initial Set of Roles}
3  foreach user u ∈ U do
4      if R(P(u)) ∉ InitRoles then
5          InitRoles ← InitRoles ∪ {R(P(u))}
6          set orig_count of R(P(u)) to 1
7      else
8          Increment orig_count of R(P(u))
9  {Subset Enumeration and User Count Computation}
10 foreach Role(i) ∈ InitRoles do
11     InitRoles ← InitRoles \ {Role(i)}
12     foreach Role(j) ∈ InitRoles do
13         if Role(i ∩ j) ∉ GenRoles then
14             Count(Role(i ∩ j)) ← Count(Role(i ∩ j)) + orig_count(i)
15             Count(Role(i ∩ j)) ← Count(Role(i ∩ j)) + orig_count(j)
16             Add i, j to the list of contributors for Role(i ∩ j)
17             GenRoles ← GenRoles ∪ {Role(i ∩ j)}
18         else
19             if i has not contributed before to Role(i ∩ j) then
20                 Count(Role(i ∩ j)) ← Count(Role(i ∩ j)) + orig_count(i)
21                 Add i to the list of contributors for Role(i ∩ j)
22             if j has not contributed before to Role(i ∩ j) then
23                 Count(Role(i ∩ j)) ← Count(Role(i ∩ j)) + orig_count(j)
24                 Add j to the list of contributors for Role(i ∩ j)
25 return GenRoles, InitRoles, Count, orin_count
```

Algorithm 3 gives the details about the first phase of Fast Miner algorithm. The first part consists of lines 3-8. The for loop iterates over all users while the if statement at line 4 either increments the count of the role (if present) or adds it to the initial set.

The second part consists of lines 10-24. The for loop in line 10 iterates over all of the roles initially created. Then the for loop in lines 12-24 intersect this set with all of the remaining roles initially created and adds all the unique intersections between pair of roles formed to *GenRoles*. If any of the two users haven't contribute to the *count* of the new role, we need to add the *count* of user in the initial set to the *count* of new roles. This is

necessary to ensure that all possible intersections take place.

Algorithm 4 Fast Miner Algorithm - Phase Two

Input: Access Matrix $D \equiv (U, P)$, $GenRoles$, $InitRoles$, $Count$, $orig\_count$
Output: $\langle R, UA, PA \rangle$

1 $R \leftarrow \emptyset, UA \leftarrow \emptyset, PA \leftarrow \emptyset$
2 {Role Prioritization}
3 Sorted $GenRoles$ by $orig\_count + Count$
4 {Generation of UA, PA}
5 **foreach** *user* $u \in U$ **do**
6 **if** $\exists\ CanditRole = \{R_i, R_{i+1}, .., R_j\} \subseteq GenRoles\ \bigwedge \bigcup_{i}^{j} R_i$ **then**
7 $UA \leftarrow UA \cup \{\langle u, \text{the } i \text{ roles}\rangle\}$
8 $R \leftarrow R\ \cup CanditRole$
9 **else**
10 **foreach** *role* $R(x) \in InitRoles$ **do**
11 **if** $x \equiv P(u)$ **then**
12 $UA \leftarrow UA \cup \{\langle u, R(x)\rangle\}$
13 $R \leftarrow R\ \cup R(x)$
14 **foreach** *role* $R(x) \in R$ **do**
15 $PA \leftarrow PA\ \cup \{\langle x, R(x)\rangle\}$
16 **return**$\langle R, UA, PA\rangle$

Algorithm 4 gives the details about the second phase of Fast Miner algorithm. The first part consists of line 3. It sorted the *GenReols* got from phase one by $orig\_count + Count$. The second part consists of lines 5-15. The for loop in line 5 iterates over all users u in the U set. First match the user u's permissions with roles from *GenRoles* set. Here we try to make the number of roles for each user as small as possible, i.e., first check if *one role* $\in$ *GenRoles* and its permission set match the user's permission set. If yes, add the user-role tuple $\langle u, role\rangle$ to UA; otherwise check if exists *two roles* $\in$ *GenRoles* and the combination of their permissions match the user's permissions. If yes, add the use-role tuple $\langle u, roles\rangle$ to the UA; otherwise continue to check if exists three roles $\in$ *GenRoles*...until the combination of all the *role* $\in$ *GenRoles* are checked. Otherwise, if the user can't match its permissions with roles from *GenRoles*, match its permissions with roles from *InitRoles*. In this thesis, we refer the experiment in [18] and set the maximum number of roles a user could have in RBAC policy generated by Fast Miner as 3.

4.1.2 Dynamic Miner(DM)

Dynamic Miner [7] introduces a new idea for prioritizing candidate roles. For role generation one can use Fast Miner or a new method based on the FP-Tree algorithm[4]. We evaluate their algorithm using Fast Miner for role generation, allowing us to more directly compare prioritization methods. That is, for role generation, we use the Fast Miner to generate the candidate role set - *GenRoles*

The main observation behind Dynamic Miner is the static prioritization used in Fast Miner does not consider candidate roles that have already been chosen, i.e., given two roles r_1 and r_2, the priority of r_2 does not depend on the creation of r_1. The set of candidate roles *GenRoles* generated in Fast Miner is often large and only a subset of the candidate roles are needed. The role selection phase selects a subset R of roles from the candidate set for the RBAC system. It takes the set of candidate roles *GenRoles* as the input and outputs a set of roles R as the roles of the RBAC system. This is the critical step in mining a flat RBAC system.

Dynamic Miner's prioritization identifies the candidate role with the highest priority r_i first, and then updates all subsequent roles under the assumption that r_i is created. For example, consider two roles r_i and r_j such that $r_i \subset r_j$. If r_j is created and no users have permissions $P_i \subset r_j$, it may not be beneficial to create r_i once r_j is created. Dynamic Miner takes at most $\min\{m, n\}$ iterations since each iteration creates a role and there are at most $\min\{m, n\}$ roles. Each iteration takes at most $n * |C|$ operations to update the "benefit" values, where C is the set of candidate roles. Therefore, the total time complexity of Dynamic Miner is $O(n * |C| * min\{m, n\})$.

We define the size of a role r (denoted as $m(r)$) as the number of permissions the role has and the support (denoted as $n(r)$) of a role as the number of users that have all permissions of the role. Vaidya et al. [18] proposed a static prioritization method for role selection. The priority of each candidate role r is $(on(r)p + n(r))$, where $on(r)$ denotes the number of users who have exactly the permissions associated with R, and p is a tunable parameter to favor roles found in the initial set of roles. We refer this approach as *Static Prioritization*. The static prioritization method is limited in the following aspects. Firstly, the priority is static in that it does not consider the choice of other roles. The user set of a role r inherits all users of its parent roles. Once a parent role is created, a subset of users in *USERS(r)* can be explained by the newly-created role and should be removed from the user set of r. Secondly, this approach does not consider the size of the role.

Algorithm 5 Role Selection of Dynamic Miner

Input: Access Matrix $D \equiv (U, P, UP)$, $GenRoles$
Output: $CanditRole$

1 $CanditRole \leftarrow \emptyset$
2 **foreach** $r \in GenRoles$ **do**
3 $\quad an(r) \leftarrow n(r)$
4 find $r \in GenReols$ *s.t.* $value = |m(r)| * |n(r)| - |m(r)| - |n(r)| = max_{r' \in GenRoles}\{|m(r)| * |n(r)| - |m(r)| - |n(r)|\}$
5 **if** $value \leq 1$ **then**
6 $\quad$ ***return*** $CanditRole$
7 $CanditRole = CanditRole \cup \{r\}$
8 **foreach** $r' \in CanditRole$ **do**
9 $\quad n(r') = n(r') - n(r') \cap an(r)$
10 $CanditRole = CanditRole \setminus r$
11 *go back to line 1*

Algorithm 5 gives the details of the algorithm. Dynamic Prioritization starts from an empty set of roles $CanditRole$, sequentially adds the current best role to $CanditRoles$, and updates the user set of other roles. For simplicity of presentation, the algorithm uses $v(r) = |m(r)| * |n(r)| - |m(r)| - |n(r)|$ to prioritize the candidate roles and will choose a role only if $v(r) \geq 1$.

If we set w_r, w_u, w_p and w_d as coefficients of the above prioritization algorithm, this approach can be generalized if different values for w_r ,w_u ,w_p and w_d are chosen. In general, the "benefit" of creating role r is $w_d * |m(r)| * |n(r)| - w_p * |m(r)| - w_u * |n(r)| - w_r$. (When $w_d = \infty$, one can use a suitably large value for w_d in the calculation.)

Each time we pick the current best role r that maximizes the above criteria and add it to R, we update the user set of other roles r since the permissions of a user can be explained by r and may not support r any more. To do this, we keep track of two user sets for each role r: the actual user set $an(r)$ and the current user set $n(r)$. The current user count $|n(r)|$ is used in evaluating the goodness of creating the role r while the actual user set $an(r)$ is used in updating the user set of other roles in $GenRoles$. This process stops when we cannot find a role with non-negative benefit, i.e., when $|m(r)| * |n(r)| - |m(r)| - |n(r)| \leq 1$ for all $r \in GenRoles$.

For the generation of UA set, as what we did in Fast Miner, we refer to the experiment in [18] and set the maximum number of roles a user could have in RBAC policy generated by Dynamic Miner as 3.

4.1.3 PairCount Miner(PC)

The PairCount Miner algorithm is based on a new idea for prioritizing roles and is presented as an alternative prioritization method for Fast Miner. It is based on the following observation. In Fast Miner's static prioritization[18], the priority of a permission set P depends on the number of exact matches and assignable users. However, in all data generation algorithms, multiple roles are assigned to a user. If almost every user is assigned more than one role, then the exact count for any original role is 0, and only using the number of assignable users does not perform well. Even when multiple roles are assigned to any user, it is possible that among all the users that share a role, there are many pairs of users that share only that role, but no other. Hence if we compute how many pairs of users share exactly P, we would obtain a high count for original roles.

Specifically, given a candidate role P, its pair count is:

$$PC(P) = |\{(u_i, u_j) | u_i \neq u_j \wedge P(u_i) \cap P(u_j) = P\}|$$

In the above equation, $P(u)$ is the permission set of a user u. We note that if a candidate roles has an exact count of n, then this will contribute $\frac{n(n-1)}{2}$ to its pair count. PC also has the same time complexity as Fast Miner, and can naturally be extended to large tuples (triples, quads, etc.).

PairCount Miner introduces a new idea for prioritizing candidate roles. Just like the Fast Miner and Dynamic Mier, for role generation one uses the approach in Fast Miner. In this thesis, we evaluate their algorithm using Fast Miner for role generation.

Algorithm 6 Enumerate and Count interesting roles of PairCount Miner

Input: Access Matrix $D \equiv (U, P, UP)$, $InitRoles$
Output: $GenRoles, Count$

```
1  GenRoles ← ∅
2  foreach Role(i) ∈ InitRoles do
3      InitRoles ← InitRoles \ {Role(i)}
4      foreach Role(j) ∈ InitRoles do
5          if Role(i ∩ j) ∉ GenRoles then
6              Count(Role(i ∩ j)) ← 1
7              Add ⟨i, j⟩ to the list of contributors for Role(i ∩ j)
8              GenRoles ← GenRoles ∪ {Role(i ∩ j)}
9          else
10             if ⟨i, j⟩ has not contributed before to Role(i ∩ j) then
11                 Count(Role(i ∩ j)) ← Count(Role(i ∩ j)) + 1
12 return GenRoles, Count
```

Algorithm 6 gives the detailed steps of the enumeration and counting interesting roles process in PairCount Miner algorithm. The for loop in line 2 iterates over all of the roles initially created. The for loop in lines 4-11 iterates this set with all of the remaining roles initially created and adds all the unique intersections between pair of roles formed to *GenRoles*. Meanwhile, add the tuple $\langle i, j \rangle$ to the list of contributors for $Role(i \cap j)$ and set the count of $Role(i \cap j)$ as 1. Otherwise, if $Role(i \cap j)$ already exists in *GenRoles* and tuple $\langle i, j \rangle$ doesn't have contribute before to $Role(i \cap j)$, increment the count of $Role(i \cap j)$. Finally after the iteration in line 2, return *Genroles* and *Count*.

For the generation of UA set, as what we did in Fast Miner, we refer the experiment in [18] and set the maximum number of roles a user could have in RBAC policy generated by PairCount Miner as 3.

4.2 Input Datasets

The majority of role mining algorithms use user-permission information as the input data. That is, the input to a role mining algorithm is an access matrix. We choose to use only user-permission information of different access matrix in this paper.

4.2.1 Datasets from the Literature

We use datasets that have been mostly used in previous role mining papers and literature. The datasets that will be used in the literature are shown in Table 4.1.

Dataset	|User|	|Perm|	|UP|
Mailer	189	678	3955
Healthcare	46	46	1486
Domino	79	231	730
EMEA	35	3046	7220
APJ	2044	1164	6841
Firewall 1	365	709	31951
Firewall 2	325	590	36428
Americas	3477	1587	105205

Table 4.1: Sizes of the real-world datasets presented

The above real word datasets were obtained from researchers at HP Labs and used for evaluation in [3]. The **Healthcare** data was from the US Veteran's Administration; the **Domino** data was from a Lotus Domino server; **Americas** (referring to americas small in [3]), **EMEA**, and **APJ** data were from Cisco firewalls used to provide external users access to HP resources. We also use their **Firewall1** and **Firewall2** policies.

4.2.2 Generated Data

The synthetic dataset was generated based on a template used in a recent paper[17]. Researchers from Stony Brook University generated a template for an RBAC system in a university setting, presumably through a process similar to top-down role engineering. They created this template for the purpose of studying security analysis in role based access control, rather than role engineering. Thus, the main consideration was to make the RBAC system as realistic as possible. Molloy et al. [9] used this template to generate a dataset for evaluation. We use that dataset as the generated dataset for the experiment.

4.3 Assumptions

Before we conduct the experiment of secrecy resilience, we make some assumption as *a priori* knowledge:

1. There exists adversary user and victim user in the access matrix. The adversary user only knows the existing *roles* and *permissions* he/her is assigned with and

the adversary always wants to know whether the victim user has those *roles* and *permissions*.

2. The probability that the victim has a role from adversary only be impacted by the structure of RBAC policy.

3. Permissions are orthogonal to one another, i.e., possession of permission p has no implications to possession of another permission q, and possession of a permission on a resource r has no implications to possession of a permission on another resource s. Hence adversary user has uniformity and independence assumptions in the access matrix, i.e., $p = \Pr\{u \text{ is assigned with permission } p\} = 1/2$ and the event that u has permission p_i and the u has permission p_j are independent for every $i \neq j$.

4. Suppose adversary user a is assigned to roles $r_1, \ldots, r_k$. Then, the only possible events for a victim user b are: (i) b is assigned no roles in $\{r_1, \ldots, r_k\}$, and, (ii) b is assigned at least one role in $\{r_1, \ldots, r_k\}$.

5. The event that victim b is assigned to at least on role in $\{r_1, \ldots, r_k\}$ follows the inclusion–exclusion principle.

Based on the above assumptions, here are several theorems we concluded and will be deployed in the calculation of secrecy resilience:

Theorem 7. For access matrix D, if we use any algorithm from User-Role Miner, Permission-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner to generate the RBAC policy and k is the number of permissions that role r has, with the assumptions we adopted, from the view of adversary user m, the probability that a victim user n possesses role r that m has is always: 2^{-k}.

Since we assume the user has uniformity and independence assumptions in the access matrix, a victim user has a role means the user has all the permissions that role has, hence the probability that a victim user n possesses role r that adversary m has is always: 2^{-k}.

4.4 Experiment Analysis

In this thesis, we make empirical results and analysis for two disclosure events:

- **Event One**: Given a RBAC policy user a who is the adversary and a role r, the event that a victim user is a member of the role r.

- **Event Two**: Given a RBAC policy user a who is the adversary, the event that a victim user is a member of some roles of which a is a member.

4.4.1 Experiment and Analysis for Event One

4.4.1.1 The Worst-Case Study

In order to get the "worst-case" for each dataset, we first generate the RBAC policy with different role mining algorithm with each of the dataset. Then, we calculate the secrecy resilience for each user in the RBAC policy as adversary. Finally, we filter the worst-case for each generated RBAC policy(The worst-case are the the smallest secrecy resilience for each of users as adversary).

Dataset	User-Role Miner	Permission-Role Miner	Fast Miner	Dynamic Miner	PairCount Miner
Mailer	3.44×10^{-13}	1	3.44×10^{-13}	3.44×10^{-13}	3.44×10^{-13}
Healthcare	6.74×10^{-13}	1	1.32×10^{-12}	1.32×10^{-12}	1.32×10^{-12}
Domino	2.54×10^{-61}	1	2.54×10^{-61}	2.54×10^{-61}	2.54×10^{-61}
EMEA	9.39×10^{-165}	1	9.39×10^{-165}	9.39×10^{-165}	9.39×10^{-165}
APJ	2.01×10^{-16}	1	2.01×10^{-16}	2.01×10^{-16}	2.01×10^{-16}
Firewall 1	1.13×10^{-183}	1	1.13×10^{-183}	1.13×10^{-183}	1.13×10^{-183}
Firewall 2	1.46×10^{-175}	1	1.46×10^{-175}	1.46×10^{-175}	1.46×10^{-175}
Americas	1.49×10^{-91}	1	1.49×10^{-91}	1.49×10^{-91}	1.49×10^{-91}
University	3.77×10^{-11}	1	3.77×10^{-11}	3.77×10^{-11}	3.77×10^{-11}

Table 4.2: Worst-Case of Secrecy Resilience for Event One

From the result, we could see that:

- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Mailer dataset is user *grr*, who has a single role which is assigned with 47 of the 678 permissions.
- An adversary that elicits the worst-case for the User-Role Miner in Healthcare dataset is user *U19*, who has a single role which is assigned with 46 of the 46 permissions. An adversary that elicits the worst-case for the Fast Miner, Dynamic Miner and PairCount Miner in Healthcare dataset is user *U10*, who has a role which is assigned with 45 of the 46 permissions.
- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Domino dataset is user *U23*, who has a role which is assigned with 209 of the 231 permissions.

- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in EMEA dataset is user *U10*, who has a role which is assigned with 554 of the 3046 permissions.
- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in APJ dataset is user *U375*, who has a role which is assigned with 58 of the 1164 permissions.
- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall1 dataset is user *U550*, who has a role which is assigned with 617 of the 709 permissions.
- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall2 dataset is user *U271*, who has a role which is assigned with 590 of the 590 permissions.
- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Americas dataset is user *U91*, who has a role which is assigned with 310 of the 1587 permissions.
- An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in University dataset is user *PPF401*, who has a role which is assigned with 40 of the 56 permissions.

4.4.1.2 The Best-Case Study

Like what we did for "worst-case", in order to get the "best-case" for each dataset, we first generate the RBAC policy with different role mining algorithm with each of the dataset. Then, we calculate the secrecy resilience for each user in the RBAC policy as adversary. Finally, we filter the best-case for each generated RBAC policy(The best-case are the the smallest secrecy resilience for each of users as adversary).

Dataset	User-Role Miner	Permission-Role Miner	Fast Miner	Dynamic Miner	PairCount Miner
Mailer	0.811	1	1	1	1
Healthcare	0.066	1	0.066	0.066	0.066
Domino	1	1	1	1	1
EMEA	0.02	1	0.02	0.02	0.02
APJ	1	1	1	1	1
Firewall 1	1	1	1	1	1
Firewall 2	0.116	1	0.116	0.116	0.116
Americas	1	1	1	1	1
University	0.811	1	0.811	0.811	0.811

Table 4.3: Best-Case of Secrecy Resilience for Event One

From the result, we could see that:

- An adversary that elicits the best-case for the User-Role Miner in Mailer dataset is user *aazad*, who has a single role which is assigned with 2 of the 678 permissions. An adversary that elicits the best-case for the Fast Miner, Dynamic Miner and PairCount Miner is user *mchaten*, which has a role with only one permission.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Healthcare is user *U7*, who has a role which is assigned with 7 of the 46 permissions.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Domino dataset is user *U15*, who has a role which is assigned with only one permission.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in EMEA dataset is user *U0*, who has a role which is assigned with 9 permissions.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in APJ dataset is user *U1002*, who has a role which is assigned with only one permission.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall1 dataset is user *U105*, who has a role which is assigned with only one permission.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall2 dataset is user *U365*, who has a role which is assigned with 6 permissions.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Americas dataset is user *U2197*, who has a role which is assigned with only one permission.
- An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in University dataset is user *FHF316*, who has a role which is assigned with 2 permissions.

Based on the results and analysis, we could get two theorems for disclosure event one:

Theorem 8. Given any access matrix with any number n of users and any number k of permissions, for disclosure event one, under our assumptions, role mining algorithms Fast, DM, Pair output the RBAC policy whose secrecy resilience is at most as high as for the policy output by Permission-Role Miner algorithm and as low as for the policy output by User-Role Miner algorithm.

Since for each user, the number of permissions of his/her role has in RBAC policy which outputted by Fast, DM, Pair Miner is no more than that of User-Role Miner and no less than that of Permission-Role Miner. So for disclosure event one, under our assumptions, role mining algorithms Fast, DM, Pair output the RBAC policy whose secrecy resilience is at most as high as for the policy output by Permission-Role Miner algorithm and as low as for the policy output by User-Role Miner algorithm.

Theorem 9. Given any access matrix with any number n of users and any number k of permissions, under our assumptions, Permission-Role Miner is the best role mining algorithm from the standpoint of secrecy resilience.

Since the role in RBAC policy generated by Permission-Role Miner always has one permission, the probability that the victim has a role from adversary is always 1/2. So that under our assumptions, the secrecy resilience of RBAC policy generated by Permission-Role Miner is always 1, which means Permission-Role Miner is the best role mining algorithm from the standpoint of secrecy resilience.

4.4.2 Experiment and Analysis for Event Two

4.4.2.1 The Worst-Case Study

In order to get the "worst-case" for each dataset, firstly, we generate the RBAC policy with different role mining algorithm with each of the dataset. Then we calculate the secrecy resilience for each user in the RBAC policy as adversary. Finally, we filter the worst-case for each generated RBAC policy.

Dataset	User-Role Miner	Permission-Role Miner	Fast Miner	Dynamic Miner	PairCount Miner
Mailer	3.44×10^{-13}	3.44×10^{-13}	3.44×10^{-13}	3.44×10^{-13}	3.44×10^{-13}
Healthcare	6.74×10^{-13}	6.74×10^{-13}	1.32×10^{-12}	1.32×10^{-12}	1.32×10^{-12}
Domino	2.54×10^{-61}	2.54×10^{-61}	2.54×10^{-61}	2.54×10^{-61}	2.54×10^{-61}
EMEA	9.39×10^{-165}	9.39×10^{-165}	9.39×10^{-165}	9.39×10^{-165}	9.39×10^{-165}
APJ	2.01×10^{-16}	2.01×10^{-16}	2.01×10^{-16}	2.01×10^{-16}	2.01×10^{-16}
Firewall 1	1.13×10^{-183}	1.13×10^{-183}	1.13×10^{-183}	1.13×10^{-183}	1.13×10^{-183}
Firewall 2	1.46×10^{-175}	1.46×10^{-175}	1.46×10^{-175}	1.46×10^{-175}	1.46×10^{-175}
Americas	1.49×10^{-91}	1.49×10^{-91}	1.49×10^{-91}	1.49×10^{-91}	1.49×10^{-91}
University	3.77×10^{-11}	3.77×10^{-11}	3.77×10^{-11}	3.77×10^{-11}	3.77×10^{-11}

Table 4.4: Worst-Case of Secrecy Resilience for Event Two

From the result, we could see that:

1. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Mailer dataset is user *grr*, who has a role which is assigned with 47 in 678 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in Mailer dataset is user *grr*, who has 47 roles and each of the role is assigned with 1 permission.

2. An adversary that elicits the worst-case for the User-Role Miner in Healthcare dataset is user *U19*, who has a role which is assigned with 46 in 46 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in Healthcare dataset is user *U19*, who has 47 roles and each role is assigned with 1 permission. An adversary that elicits the worst-case for the Fast Miner, Dynamic Miner and PairCount Miner in Healthcare dataset is user *U19*, who has 2 roles and one of which has 31 permissions and the other has 45 permissions.

3. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Domino dataset is user *U23*, who has a role which is assigned with 209 in 231 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in Domino dataset is user *U23*, who has 209 roles and each role is assigned with 1 permission.

4. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in EMEA dataset is user *U10*, who has a role which is assigned with 554 in 3046 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in EMEA dataset is user *U10*, who has 554 roles and each role is assigned with 1 permission.

5. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in APJ dataset is user *U375*, who has a role which is

assigned with 58 in 1164 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in APJ dataset is user *U375*, who has 58 roles and each role is assigned with 1 permission.

6. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall1 dataset is user *U550*, who has a role which is assigned with 617 in 709 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in Firewall1 dataset is user *U550*, who has 617 roles and each role is assigned with 1 permission.

7. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall2 dataset is user *U271*, who has a role which is assigned with 590 in 590 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in Firewall2 dataset is user *U271*, who has 590 roles and each role is assigned with 1 permission.

8. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Americas dataset is user *U91*, who has a role which is assigned with 310 in 1587 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in Firewall2 dataset is user *U91*, who has 310 roles and each role is assigned with 1 permission.

9. An adversary that elicits the worst-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in University dataset is user *PPF401*, who has a role which is assigned with 40 in 56 permissions. An adversary that elicits the worst-case for the Permission-Role Miner in University dataset is user *PPF401*, who has 40 roles and each role is assigned with 1 permission.

4.4.2.2 The Best-Case Study

Like what we did for "worst-case", in order to get the "best-case" for each dataset, firstly, we generate the RBAC policy with different role mining algorithm with each of the dataset. Then we calculate the secrecy resilience for each user in the RBAC policy as adversary. Finally, we filter the best-case for each generated RBAC policy.

Dataset	User-Role Miner	Permission-Role Miner	Fast Miner	Dynamic Miner	PairCount Miner
Mailer	0.811	0.811	0.954	0.811	0.954
Healthcare	0.0659	0.0659	0.0659	0.0659	0.0659
Domino	1	1	1	1	1
EMEA	0.02	0.02	0.02	0.02	0.02
APJ	1	1	1	1	1
Firewall 1	1	1	1	1	1
Firewall 2	0.116	0.116	0.116	0.116	0.116
Americas	1	1	1	1	1
University	0.814	0.814	0.814	0.814	0.814

Table 4.5: Best-Case of Secrecy Resilience for Event Two

From the result, we could see that:

1. An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Mailer dataset is user *ahanna*, who has a role which is assigned with 2 in 678 permissions. An adversary that elicits the best-case for the Permission-Role Miner, in Mailer dataset is user *ahanna*, who has 2 roles and each of the role is assigned with 1 permission.

2. An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Healthcare dataset is user *U17*, who has a role which is assigned with 7 in 46 permissions. An adversary that elicits the best-case for the Permission-Role Miner in Healthcare dataset is user *U17*, who has 7 roles and each of the role is assigned with 1 permission.

3. An adversary that elicits the best-case for the User-Role Miner, Permission-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Domino dataset is user *U15*, who has a role which is assigned with 1 permission.

4. An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in EMEA dataset is user *U0*, who has a role which is assigned with 9 permissions. An adversary that elicits the best-case for the Permission-Role Miner in EMEA dataset is user *U0*, who has 9 role and each of the role is assigned with 1 permissions.

5. An adversary that elicits the best-case for the User-Role Miner, Permission-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in APJ dataset is user *U1002*, who has a role which is assigned with 1 permissions.

6. An adversary that elicits the best-case for the User-Role Miner, Permission-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall1 dataset is user *U105*, who has a role which is assigned with 1 permission.

7. An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Firewall2 dataset is user *U365*, who has a role which is assigned with 6 permissions. An adversary that elicits the best-case for the Permission-Role Miner in Firewall2 dataset is user *U365*, who has 6 roles and each of them is assigned with 1 permission.

8. An adversary that elicits the best-case for the User-Role Miner, Permission-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in Americas dataset is user *U2197*, who has a role which is assigned with 1 permission.

9. An adversary that elicits the best-case for the User-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner in University dataset is user *FHF316*, who has a role which is assigned with 2 permissions. An adversary that elicits the best-case for the Permission-Role Miner in University dataset is user *FHF316*, who has 2 roles and each of them is assigned with 1 permission.

Chapter 5

Conclusion and Future Work

In this thesis, we propose and introduce the notion of secrecy resilience as some measure of resilience that an authorization policy has to discovery of components of it. Then we implement five role mining algorithms (User-Role Miner, Permission-Role Miner, Fast Miner, Dynamic Miner and PairCount Miner) and generate the corresponding RBAC policy from both generated dataset and datasets from literature. Finally, we conduct experiment for the performance of secrecy resilience for the generated RBAC policy, and analyze the result with the best-case and worst-case of secrecy resilience.

There are amounts of topics and research areas for future work. First with a more powerful server, the maximum number of roles a user could have in the RBAC policy could be increased from 3 to a bigger number and the experiment could be conducted with the improved RBAC policy. Secondly, in this thesis, the building of notion of secrecy resilience and the role mining algorithms we used all fall in RBAC policy with a flat structure. In the future, we will set up the notion of secrecy resilience for RBAC policy with hierarchy.

References

[1] Bill Ballad. *Access Control, Authentication and Public Key Infrastructure*. Place of publication not identified Jones Bartlett Publishers Incorporated, 2010.

[2] D. Chadwick, W. Xu, and S. Otenko. Muti-session separation of duties (msod) for rbac. In *Proceedings of 23rd International Conference on Data Engineering Workshop*, pages 744–753, 2007.

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

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *In Proc. ACM International Conference on Management of Data (SIGMOD)*, pages 1–12, 2000.

[5] J. Joshi, E. Bertino, and U. Latif. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, pages 17(1): 4–23, 2005.

[6] J. Joshi, B. Shafiq, and A. Ghafoor. Dependencies and separation of duty constraints in gtrbac. In *Proceedings of the 8th ACM symposium on Access control models and technologies*, pages 51–64, 2003.

[7] Ninghui Li, Tiancheng Li, Ian Molloy, Qihua Wang, and Elisa Bertino. Role mining for engineering and optimizing role based access control systems. *Technical Report 2007-60, CERIAS, Purdue University*, pages 1–12, 2007.

[8] Gilbert MD, Lynch N, and Ferraiolo FD. An examination of federal and commercial access control policy needs. In *National Computer Security Conference*, page 107, 1995.

[9] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with semantic meanings. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 21–30, 2008.

[10] Ian Molloy, Ninghui Li andTiancheng Li, Ziqing Mao, Qihua Wang, and Jorge Lobo. Evaluating role mining algorithms. *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 95–104, 2009.

[11] P.Marikkannu, J.J. Adri Jovin, and T.Purusothaman. Fault-tolerant adaptive mobile agent system using dynamic role based access control. *International Journal of Computer Applications*, page 20 (2): 1–6, 2011.

[12] Agrawal R., J. Kiernan, R. Srikant, and Y Xu. Hippocratic databases. *In: Proc. 28th Int'l Conf. on Very Large Data Bases*, pages 143–154, 2002.

[13] Gregory Saunders, Michael Hitchens, and Vijay Varadharajan. Role-based access control and the access control matrix. *Information and Communications Security*, pages 145–157, 2003.

[14] J. Schlegelmilch and U. Steffens. Role mining with orca. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 168–176, 2005.

[15] J. Schlegelmilch and U. Steffens. Role mining with orca. *In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 168–176, 2005.

[16] Claude Elwood Shannon and Weaver Warren. The mathematical theory of communication. *Urbana : University of Illinois Press*, pages 11–55, 1949.

[17] S. D. Stoller, P. Yang, C. R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. *14th ACM Conference on Computer and Communications Security 2007*, pages 445–455, 2007.

[18] J. Vaidya, V. Atluri, and J. Warner. Roleminer: Mining roles using subset enumeration. In *In Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 144–153, 2006.

[19] W. Yamazaki and H. Hiraishi. Designing an agent based rbac system dynamic security policy enabling technologies: Infrastructure for collaborative enterprises. In *The 13th IEEE International Workshops on Enabling Technologies*, pages 199–204, 2004.

[20] Dong Yan, Zhengqiu Yang, and Chen Liu. An improved rbac96 model with range restricted. In *2009 International Forum on Information Technology and Applications*, pages 591–594, 2009-05, Vol.3.

[21] D. Zhang, K. Ramamohanarao, and T. Ebringer. Role engineering using graph optimisation. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 139–144, 2007.