FEASIBILITY ANALYSIS OF ACCESS CONTROL POLICY MINING

by

SHUVRA CHAKRABORTY, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

COMMITTEE MEMBERS:
Ravi Sandhu, Ph.D., Chair
Palden Lama, Ph.D.
Wei Wang, Ph.D.
Xiaoyin Wang, Ph.D.
Ram Krishnan, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
December 2021

DEDICATION

I would like to dedicate this work to my parents, Kshitish Chandra Chakraborty and Shipra Chakraborty. Also, I would like to dedicate it to my beloved life partner Subrata Bhadra for his continuous support to complete this PhD journey.

ACKNOWLEDGEMENTS

My PhD journey was an unforgettable learning experience for me. It was completely different from the other degrees I earned in my life and I would like to thank many people for this. First, I would like to thank my supervisor Dr. Ravi Sandhu; I might not be here today without his guidance. It is such an incredible experience to work under a mentor like him. I get to learn something new from him in every meeting. I would like to thank him for showing the correct path to shape my research, leading me with his vast knowledge, and encouraging as well as supporting me to attend conferences for showcasing my work.

I would like to thank Dr. Ram Krishnan for his invaluable guidance and advice which really helped me to shape my research goals since I joined. I would also like to thank my PhD committee members Dr. Palden Lama, Dr. Xiaoyin Wang, and Dr. Wei Wang for their time, knowledge, and insightful comments to organize the dissertation.

I appreciate the support I have got from Suzanne Tanaka and James Benson from ICS. I would like to say thanks to Susan Allen from CS department for answering so many queries patiently. Finally, I would like to acknowledge all the faculty and stuff members of the Computer Science department at UTSA for their wisdom and support. I am grateful to all my friends, family members, and well wishers; I could not accomplish this without you all.

This Masters Thesis/Recital Document or Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Masters Thesis/Recital Document or Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Masters Thesis/Recital Document or Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Masters Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Masters Thesis/Recital Document or Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript

describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Masters Thesis/Recital Document or Doctoral Dissertation attest to the accuracy of this statement.

December 2021

FEASIBILITY ANALYSIS OF ACCESS CONTROL POLICY MINING

Shuvra Chakraborty, Ph.D.
The University of Texas at San Antonio, 2021

Supervising Professor: Ravi Sandhu, Ph.D.

Access control enforces who can access what inside a system, allowing only legitimate users to get legitimate access to resources inside the system. To clarify, access control governs resource access based on a variety of criteria, such as user credential verification, environmental conditions, resource characteristics, and so on. To keep up with the fast changing requirements, new "robust and resilient" models in the access control domain are being developed to keep pace with the expanding complexity and innovation of technology, such as ABAC (Attribute-Based Access Control), ReBAC (Relationship-Based Access Control), AReBAC (Atrribute-aware ReBAC), etc. When a system is already protected by an established access control model, the "policy mining problem" refers to the process of automating or at least partially automating the conversion to another model. To migrate to another target access control system, policy mining generally requires the existing source access control model and additional information. Policy mining tasks are frequently guided by a set of assumptions, such as the target access control system must have the identical set of users, resources and authorizations. Our investigation begins in pursuit of the feasibility of access control policy mining under specific assumptions, which is essentially an in-depth examination of various types of policy mining issues.

This dissertation investigates feasibility analysis of access control policy mining for variety of source and target access control models, develops algorithms to find the feasibility, resolves the cases of infeasibility, and demonstrates effectiveness of the developed approaches through mathematical proofs and implementation.

The first step towards feasibility analysis begins with ABAC policy mining, where the source access control system was Enumerated Authorization System (EAS). Using the limitations of the state-of-the-art ABAC policy mining approaches, it first develops the concept of feasibility in

ABAC policy mining, formally named as ABAC RuleSet Existence Problem. Furthermore, the concept of feasibility in ABAC policy mining was extended while source access control model is Role Based Access Control (RBAC) system as well. Besides, for both cases, infeasibility solution algorithms are proposed and unrepresented partition problem is also discussed.

Feasibility of ReBAC policy mining was explored in the second step. Similar to the first step, feasibility problem in ReBAC policy mining is defined and formulated as ReBAC RuleSet Existence Problem. In addition, different versions of ReBAC RuleSet Existence Problem are introduced. Infeasibility solution, and significant directions for future enhancement are noted. Significant example cases are included to demonstrate the effectiveness of the proposed approach.

Although feasibility analysis of ABAC and ReBAC policy mining provide an insightful study, however, the combination of both, Attribute-aware ReBAC increases the expressiveness and flexibility. Attribute-aware ReBAC RuleSet Existence Problem is introduced in this context, analyzed and feasibility as well as infeasibility algorithms are provided with associated proofs. One important contribution here is: the notion of approximate solutions in the case of infeasibility is briefly mentioned. Later, significant example cases are discussed with future directions.

As the final step, this dissertation introduces a novel concept of extending the concept the feasibility analysis of ABAC and ReBAC policy mining, formally named as Extended ABAC and ReBAC RuleSet Existence Problem, EAREP and ERREP in short, respectively. Initially, the motivation and objective of defining these problems are demonstrated with example. Later, infeasibility solution and associated pros and cons are discussed briefly.

Finally, the dissertation work is concluded with significant directions for future extensions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Protecting information or other resources from unauthorized access is one of the prime components in security enforcement, where access control comes into play. Access control coexists with other security services in a system and is a vital part of ensuring that resources or objects get accessed by legitimate users only. Numerous access control models have been proposed so far, beginning from Lampson's access matrix to date; however, only a couple of them achieved the practical traction to be widely recognized. These include, Discretionary Access Control (DAC) [39], Mandatory Access Control (MAC, also referred as Lattice-Based Access Control) [40], Role-Based Access Control (RBAC) [21, 38], Attribute-Based Access Control (ABAC) [24, 28], Relationship-Based Access Control (ReBAC) [22, 23].

DAC is a classical model that became widely popular due to its flexibility and simplicity. In DAC, access to resources is verified based on the discretion of the resource/object owner. MAC overcomes a key limitation of DAC: it imposes restrictions on the flow of information from higher to lower level. Usually, each object and user in MAC is assigned a security label, expressing their sensitivity level in the system. A user can access an object only if some relationship between their security labels holds. The general assumption is: higher security label dominates over lower ones, according to the hierarchy of labels established by the system policy. As a matter of growing demand in many commercial enterprise system, there have been a good amount of works to extend DAC and MAC but a more organized access control model was needed to overcome the rigid characteristics of MAC and the autonomous nature of DAC. RBAC successfully filled this gap when it came into picture in the early'90s. The main component of RBAC is role, an intermediary between user and permissions in the system. In basic RBAC, a user can have multiple roles where each role might contain multiple permissions assigned to it [21]. Although RBAC is a widely popular model for its capability to manage large scale authorizations, it has limitations which Attribute-Based Access Control (ABAC) seeks to address. ABAC can encompass the advantages as well as go beyond the limits [28] of all the foregoing three models mentioned so far. Although it is

arguable which one of RBAC or ABAC is more flexible, scalable, auditable, and provides more support for the dynamic environment [44], the emergence of ABAC cannot be denied. ABAC can be configured to DAC, MAC, and RBAC [28], suitable for large enterprises and notably overcome some limitations of RBAC, for example, role explosion [25]. The flexibility of ABAC in dynamic as well as distributive environment causes migration to ABAC from different already deployed access control models. Another well-recognized field of access control model is Online social networks (OSNs) which is different from traditional ones. OSNs have emerged rapidly over the past several years and now, billions of users in all over the world are connected through different relationships! Relationship-Based Access Control (ReBAC) is employed to protect the huge amount of sensitive and private information (i.e., photo, contacts, blogs) over the OSNs from unauthorized manipulation. Although ReBAC expresses authorization through direct and indirect relationships, there are cases where only using relationships is insufficient. Consider a policy that, in a social network, one can send a friend request to anyone older than her. Here, age of source and target must be known. So, each user in that social network must have an attribute/characteristic indicating age. Integrating attributes with ReBAC components certainly add more expressiveness [18], formally named as Attribute-aware ReBAC (AReBAC).

Now, one question is, which one is the best to use? Depending on the system requirement, access control models can sustain throughout the changes made by the policymakers. However, sometimes it gets necessary to migrate from one access control system to another. For example, suppose an organization uses the Access Control List (ACL) for system resource protection purposes. Due to the growing size of the organization and change in the mode of operation, now it is being difficult to manage the huge set of permitted authorizations. Therefore, system administrator decided to migrate to RBAC or ABAC (advantages of ABAC and RBAC will be discussed later). Although it is possible to complete the migration process by relying on human support, an automated solution that is free of manual errors may require less effort and time. Access control policy mining refers to the partially automated migration process from one access control system to another. The feasibility analysis of access control policy mining is the general term that encom-

passes all those issues that arise during the migration. For example, as mentioned earlier, given an ACL (access control list) and supporting attribute data, what could be the list of few possible issues arise during the migration process to ABAC?

1. Are those given data sufficient enough to generate an equivalent ABAC system? (By equivalent, we mean the exact set of authorizations allowed in both systems where inexact or approximate equivalency means the more or less number of authorizations allowed.)

2. If the given data is insufficient, how to generate approximate solutions with suitable security measure?

3. Can the given attribute set be minimized/reduced? Does that reduction process hamper the expressiveness of ABAC?

4. Is it possible to design the suitable attribute set so that migration can be fully automated?

5. Is it possible to reduce the generated policy size even if the generated system is equivalent?

Considering the variety of issues, domain of feasibility analysis in Access Control Policy Mining merits systematic study. The domain of the feasibility of access control policy mining includes all mining problems such as RBAC mining [33], ABAC mining [41], AReBAC mining [6] and so on. Hence, it can be stated that, as new access control model joins the access control domain, the feasibility analysis domain also grows accordingly.

This dissertation focuses on the formulation, development, and implementation of feasibility analysis in access control policy mining domain, especially for ABAC, ReBAC and AReBAC policy mining problems. Rigorous examples of motivation and detailed case studies are also provided. Based on the analysis works so far, classification of infeasibility solutions, and future enhancement are also noted. It should be noted that, each type of feasibility analysis formulates their own access control model specification while utilizing the fundamental aspects of model itself.

1.1 Problem Statement

While access control policy mining problems are an emerging research concern, the issues which arise during the automated migration process (such as use of entity ids in the policy, the equivalence of the source and target access control systems) are not yet addressed rigorously. There is a significant gap, especially lack of formal logical frameworks and academic literature with respect to variety of access control policy mining problems. On the other hand, characteristics of access control models, supplementary data required for the migration process and the amount of efforts required to automate the whole process make the feasibility analysis technically interesting. Thereby, a conceptual framework needs to be defined in order to facilitate a smooth migration from a source access control system to the target so that it cope up with the real world access control policy mining applications.

1.1.1 Thesis Statement

As a matter of growing real-world challenges and advancements in technology, migration of one access control system to another is an emerging problem. The complete or partially automated solution to this migration process is called the access control policy mining problem. During the mining process, a set of assumptions and criteria are imposed to precisely define the migration goals. The feasibility analysis of the access control policy mining problem formulates the logical framework of the problem, resolves the infeasibility issues possibly arising during the policy mining process so that the solution can satisfy those imposed criteria, and provides a rigorous foundation for the migration process.

1.2 Scope and Assumption

Briefly, the scope and assumptions of the proposed work are as follows:

1. In this dissertation, the domain of feasibility analysis of the access control mining problem is comprised of the domain of access control models. Here, we work with the best known

models (RBAC, ABAC, ReBAC, etc.).

2. As the feasibility analysis framework is newly defined, infeasibility solution approaches do not follow any specific guidelines, and performance measurement is limited to analyzing algorithmic complexity only. It can be said, use of real world data would make the experimental results more interesting.

3. Right at this moment, any issue related to the feasibility of migration of one access control system to another is part of our problem domain. However, a clear boundary is yet to be defined. For example, if we migrate an ACL system to RBAC, having the equivalent protection from the generated system could be the determining factor. Besides, getting the optimal role set even if the equivalency has to be compromised; is also a part of the feasibility analysis domain.

4. RBAC, ABAC, and other access control model (Which models are the key component of our work) are defined and extended by many previous works and vary from each others in features. Here, we work with our own version of those access control models such as [13,14]. Our work is not directly capable of providing solutions for all situations. A separate study is needed to extend our work in these areas.

5. This dissertation work does not claim to compete with human expertise at all. Rather than that, this dissertation focuses on the full automation of migration process while a certain set of instruction or criteria are imposed.

1.3 Summary of Contribution

The major contributions of this dissertation are visually presented in Fig. 1.1 and listed below.

1. Established a novel direction for defining feasibility analysis in access control policy mining domain.

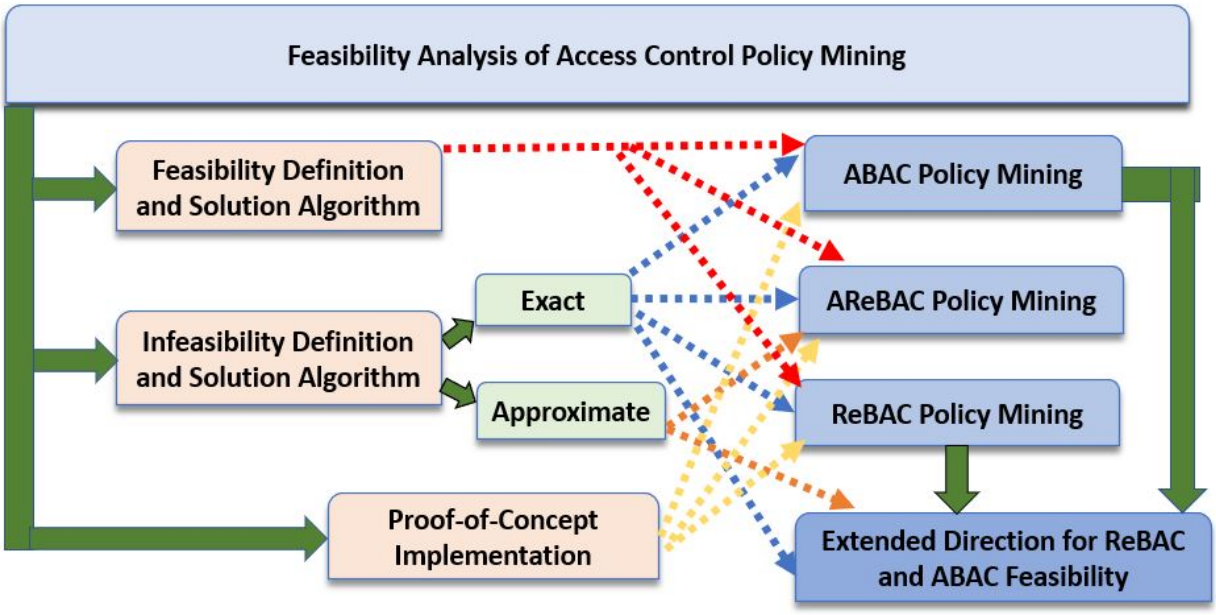

Figure 1.1: Summary of contribution

2. Developed algorithms for feasibility analysis for a particular set of access control mining problems such as ABAC and ReBAC.

3. In the case of infeasibility, solution algorithms are presented to make it feasible under given criteria. The solutions are mostly equivalent, however, approximate solutions are also discussed in some cases.

4. Showed usefulness of feasibility analysis process through a qualitative measure such as complexity analysis.

5. Demonstrated the generated algorithms with proof-of-concept case studies to show the effectiveness.

1.4 Organization of the Dissertation

The rest of this dissertation is organized as follows: chapter 2 presents an introductory review on popular access models, and policy mining literatures. In chapter 3, the work on the feasibility of ABAC policy mining problem from authorizations and RBAC are presented. Chapter 4 discusses

the work on the feasibility of ReBAC policy mining along with infeasibility solutions. Chapter 5 formulates feasibility of AReBAC policy mining where inexact solution of infeasibility problem is introduced as well. Chapter 6 proposes an extension towards the feasibility of ReBAC policy mining, and finally, Chapter 7 summarizes the dissertation work and lists some interesting direction for future works.

CHAPTER 2: LITERATURE REVIEW

In this chapter, a brief overview of related access control models is presented. It further discusses prior literature on the theory of access control policy mining. Although feasibility of access control policy mining encompasses the entire domain of access control models, this dissertation work is limited to the most relevant ones, such as RBAC, ABAC. ReBAC, and AReBAC.
Significant portion of this chapter has been published as background studies at the following venues [11–14].

- Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. 2019. On the Feasibility of Attribute-Based Access Control Policy Mining. In 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI). 245–252.

- Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. 2020. On the Feasibility of RBAC to ABAC Policy Mining: A Formal Analysis. In Secure Knowledge Management In Artificial Intelligence Era. Springer Singapore, Singapore, 147–163.

- Shuvra Chakraborty and Ravi Sandhu, Formal Analysis of ReBAC Policy Mining Feasibility. In Proceedings of the 11th ACM Conference on Data and Application Security and Privacy (CODASPY), Virtual Event, April 26-28, 2021.

- Shuvra Chakraborty and Ravi Sandhu, On Feasibility of Attribute-Aware Relationship-Based Access Control Policy Mining. In Proc. 33rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec), Virtual Event, July 19-20, 2021.

2.1 Access Control Models

The main purpose of access control is to limit the activities of the legitimate users in the system where it is presumed that the identity of the user is pre-verified. Access control is not the only one that protects the system from security breaches; rather it is coupled with other security measures like authentication, auditing, etc. The very basic components of an access control model are

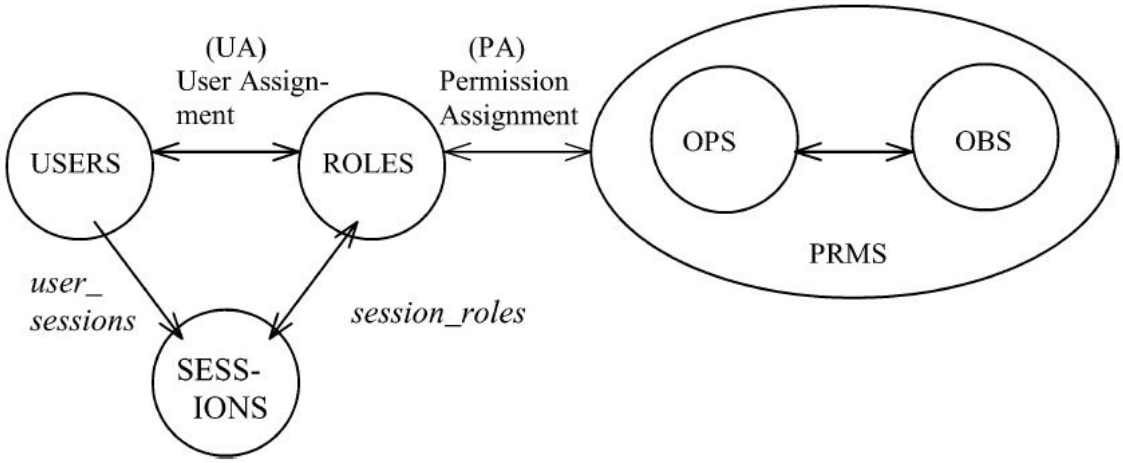

Figure 2.1: The Core RBAC model [21]

user/subject and object/resources. Although some access control models make a clear distinction between user and subject [28], in our work, user and subject are used interchangeably.

2.1.1 Role-Based Access Control (RBAC)

The base component of RBAC [21, 38] is the role, an intermediary between user and permission in the system. Each user in the system acquires the set of permissions through the roles being assigned to them. Usually, there is a system administrator who assigns the permissions to the roles, although the role and permission assignment setup may vary depending on the model and other factors. The NIST standard of RBAC [21] comprises of these four: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations. Amongst them, Core RBAC model with five basic elements (users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS)) is shown in Fig. 2.1.

RBAC is policy-neutral [38], auditable, offers permission and user-level abstraction, and provides operational and administrative scalability through roles.

2.1.2 Attribute-Based Access Control (ABAC)

RBAC has been a de facto standard access control model for over the last two decades. However, some limitations of RBAC have drawn practitioner attention into possible extensions of RBAC or invention of a new model. Among the RBAC limitations, a couple of notable issues are: i) in dynamic environments, additional information other than role might be required to evaluate access decisions, for example, an employee can access office files while his official device is connected to the internet through office LAN but not from home/other places so that apart from the employee role, that particular device location and time is important here, ii) Role explosion problem [20], and iii) it is hard to manage homogenous objects with Basic RBAC which does not support object attributes, etc. ABAC came into the picture to overcome the shortcomings of other access control models. ABAC is flexible [24, 25], able to work as an alternative of DAC, MAC, and RBAC [28], avoids the necessity for explicit authorizations to be directly assigned to individual subjects prior to an access request [25]. Fig. 2.2 shows a unified ABAC model [28] for reference purpose; the prime components are users (U), subjects (S), objects (O), user attributes (UA), subject attributes (SA), object attributes (OA), permissions (P), authorization policies, and constraint checking policies for creating and modifying subject and object attributes. In our work, we have avoided the explicit distinction between user and subject, more details are available in [13, 14].

2.1.3 Relationship Based Access Control (ReBAC)

So far, we have discussed access control models which evaluate access decisions depending on the identities, sensitivity level, role, and attributes. ReBAC is a bit different from those models mentioned earlier (DAC, MAC, RBAC, and ABAC) and is especially used for online social networks (OSN) [18]. OSN requires an access control system that offers support to scalability and dynamicity, lets the user and resource owner specify their own policy and evaluates access requests (hence provide security and privacy) based on the relationship. The relationship in OSN could be of different types, user to user [16], object to object [1] and so on. In [4], ReBAC is represented as an object-oriented extension of ABAC where entities (user, resource, etc.) are referred as classes and

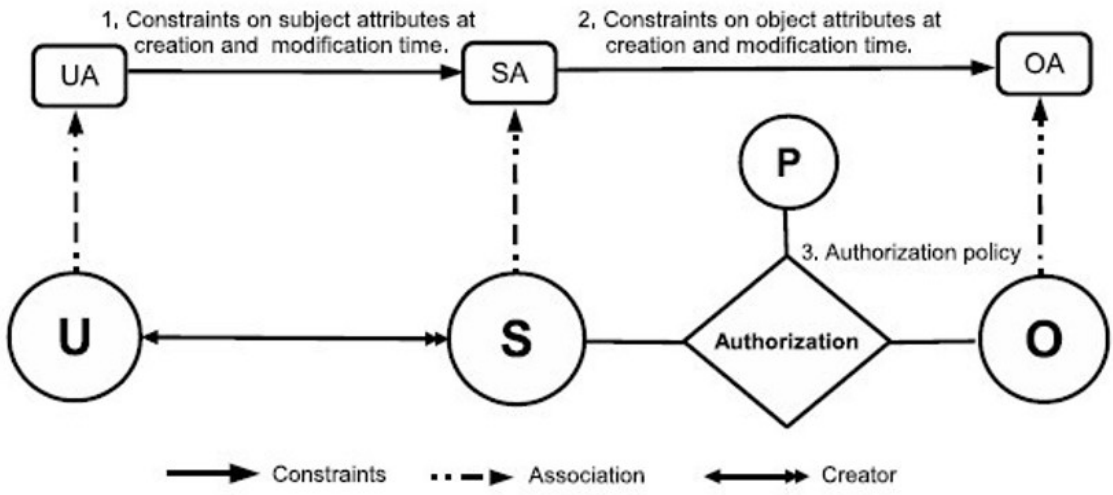

Figure 2.2: $ABAC_\alpha$ model components [28]

relationships are expressed as fields between classes. For example, Fig. 2.3 shows an example of a class model diagram in [4]. In this example, each of the rectangular boxes represents an entity (or class) in the system where the directed arrows from one class to another denote that the originator has a field (relationship) of the pointed class. For more details, [4] can be consulted.

2.1.4 Attribute-aware Relationship-Based Access Control

Although ReBAC offers an effective way of configuring access control policies in OSNs, there are shortcomings as well. For example, most ReBAC models cannot exploit the complicated topological information residing in a social graph, other than type, depth, or strength of relationships [18]. Besides, it generally lacks support for different contextual information of users, resources, and relationships available in OSN (also called attribute, utilized for flexible and finer grained access control) [18]. Attribute-aware ReBAC incorporates variety of attributes with ReBAC structure, such as [18] mentions about i) node (usually, node in a social graph is user or resource), ii) edge (relationship), and iii) count (the occurrence requirement for the attribute-based path) attributes. In [18], an Attribute-aware ReBAC policy has been specified which enhances access control capability and offers finer-grained controls compared to ReBAC. [37] presents an attribute-supporting ReBAC model for Neo4j (a popular graph database) that provides finer-grained access control by

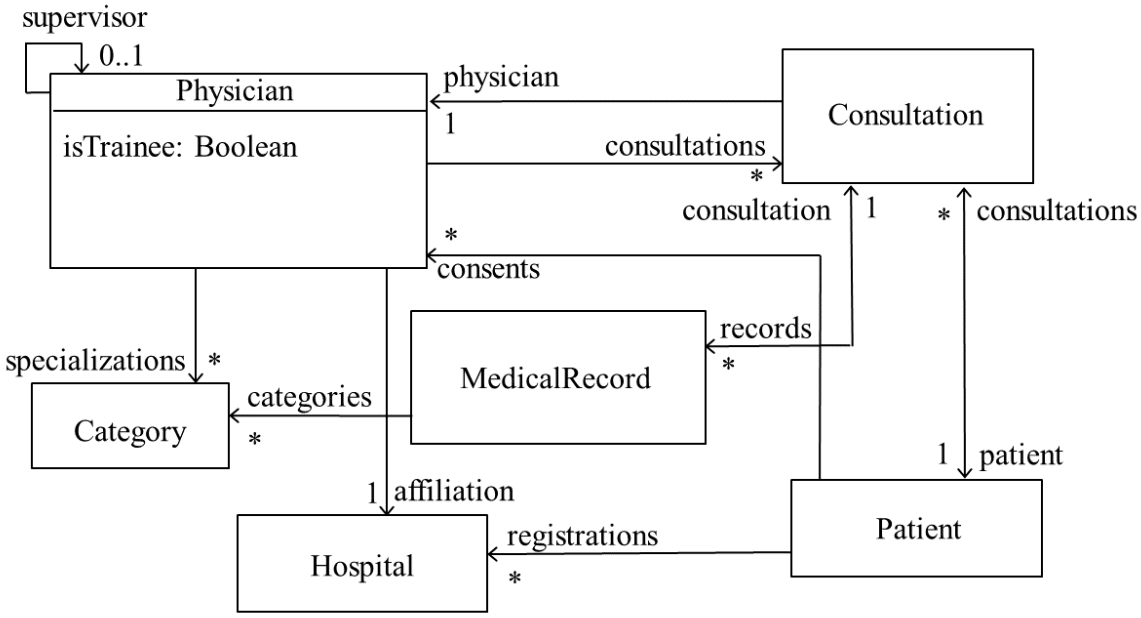

Figure 2.3: ReBAC class diagram example from [4]

operating over resources.

2.2 Background of Feasibility Analysis

Each access control system has its pros and cons and is chosen according to system requirements by the administrator. However, advancements in technology, change in mode of operations and growing/shrinking size of the organization may lead to migration to another more appropriate access control model. Therefore, access control policy mining comes into the field to simplify/partially automate the conversion process. Manual effort is often error-prone, time consuming and experience-based, That's why, a robust, reliable, and efficient (in terms of resource utilization, such as memory or runtime) access control policy mining technique may reduce the time, effort needed, and expense related to transition from one access control system to another.

Based on the circumstances stated above, automation of access control policy mining can certainly reduce the human effort and time needed, remove the burden of checking errors. Feasibility analysis answers the question related to access control policy mining such as i) is the support-

ing data good enough under the given criteria, ii) can we replace/remove the supporting data to simplify the migration process, and so on. As far as we could possibly know, feasibility notions of access control policy mining problem has been introduced without precedent in this dissertation. There is no prior literature that deals explicitly with these issues. There exist noteworthy past works on the field of access control policy mining, such as rule mining, role mining, ABAC mining, Relationship-Based Access Control (ReBAC) mining [4], etc.

2.2.1 Access Control Policy Mining Review

There are notable previous works on the field of policy mining, such as rule mining, role mining, ABAC mining, Relationship-Based Access Control (ReBAC) mining [4], and so on. For example, Role Mining [33, 35, 36, 43, 48] is a prevalent branch of RBAC policy mining which partially automates the process "role" construction, while user-permission assignment and possibly other information are given. In recent years, Attribute Based Access Control (ABAC) policy mining has been studied, whose feasibility is in the scope of this dissertation.

ABAC policy mining problem was first introduced formally in [47]. Given an access control list policy as input, [47] finds out equivalent ABAC policy. Here, ABAC rule is a tuple specifying sets of users, objects, operations, and constraints involving user and objects attributes. Although constraints give more generalized rule, but only a few forms of constraints are allowed in this study. Another work with authorization data as input is [42]. Despite having the same asymptotic complexity, [42] shows better performance with respect to total execution time. In [42], two algorithms for ABAC mining, ABAC-FDM, and ABAC-SRM has been proposed. ABAC-FDM is accurate but due to its exponential complexity, more efficient ABAC-SRM is proposed. Both [42, 47] aim at generating compact ABAC policy, more specifically, set of ABAC rules with minimum Weighted Structural Complexity(WSC) as described in [47] and deal with positive ABAC rules only.

A new out of the box approach is given in [26], which deals with positive as well as negative ABAC rules. This work basically depends on PRISM, an existing rule mining algorithm. A special note about ABAC policy mining algorithm in [26] is, a complete log is assumed to be given as input

or denied otherwise. The output is consistent with respect to input log like previous works.

Based on variety of the input data, some other notable ABAC policy mining works are: from RBAC [45], log data [46], sparse log [19], etc. A deep learning approach towards ABAC policy mining from logs using Restricted Boltzmann Machine (RBM) has been presented in [34]. Apart from these works, an evolutionary computation approach for ABAC policy mining is presented in [32], based on incremental learning of single rules and search-optimizing features. An unsupervised learning based approach for mining ABAC policies is described in [29]. Another informative literature can be found in [30].

The recent proliferation of OSNs has accelerated the research of finding an access control paradigm which is different from traditional dominant access control models like ABAC [25], RBAC [21], etc. According to the early literature, ReBAC policy is characterized by the explicit tracking of interpersonal relationships between users [22]. ReBAC is a general-purpose access control model which supports the natural expression of parameterized roles, the composition of policies, and the delegation of trust [23]. A further extended hybrid-logic based ReBAC policy is given in [3].

In general, given an OSN, users and resources are interconnected via various types of relationships. In order to specify ReBAC policies, particular relationship directions between users and resources can play significant roles. For example, [17] specifies ReBAC policies based on user to user (U-U) relationships in OSN. Similarly, [1] uses resource to resource (R-R), and [15] uses resource to user (R-U) and vice versa to express ReBAC policies. In addition, [2] presents a comparative analysis of expressive power and performance implications between ReBAC and ABAC features, [31] does an extensive analysis when the OSN is updated, and [18] proposes ReBAC to be integrated with ABAC to enhance the capability and allows finer-grained controls.

Given an access control system along with supporting data, ReBAC policy mining algorithms find the equivalent ReBAC policy. This provides partial automation to the overall migration process, reduces cost and uses some measures to find the most efficient rule set. A few works on ReBAC policy mining are discussed briefly as follows. The work in [8] presents ReBAC as an

object-oriented extension of ABAC where the "class" structure is able to realize the relationship between various entities, beyond the user and resources paradigms. In [9], the work in [8] is basically extended, where heuristic-guided greedy and grammar-based evolutionary algorithms for ReBAC policy mining are presented. A further extension is proposed in [7], to the evolutionary ReBAC policy mining in [9]. The extended ReBAC policy mining in [7] follows the simplification as well as feature selection by using neural network resulting in a more scalable and efficient algorithm. Some other ReBAC policy mining algorithms use decision tree [5], incomplete and noisy input data [10], and mine ReBAC policies from graph transition [27]. In comparison with [5, 10, 27], this feasibility study is limited to static relationship graph with complete input information only.

Compared to [5, 7–10], our work in this study concentrates on whether ReBAC mining is feasible or not without altering the core spirit of ReBAC, i.e., relationships should be the key to express policies and use of unique user or resource ids is prohibited. This is a fundamental difference since ReBAC policy mining is always feasible with such ids.

Although both ABAC and ReBAC have their own advantages to express authorization policies (e.g., [2] presents a rigorous study on that), integrating ABAC with ReBAC can provide finer-grained controls and improve the expressiveness that is not present in standalone ABAC or ReBAC. For example, [18] presents an attribute aware ReBAC access control model.

Although the policy specification language in this study is far different from [6, 37], these two works should be recognized as related works. In [6], an approach to mine ABAC and ReBAC policies has been proposed where access control lists and incomplete information about entities are given. A few significant points about [6] are i) the proposed algorithm prefers the context of ReBAC mining because ReBAC is more general than ABAC, ii) entity ids are allowed to be used (which makes the generated policy less general), and iii) there is a policy quality metric available. Compared to [6], entity ids are strictly prohibited in the attribute-aware context of this study. On the other hand, [37] presents an attribute-supporting ReBAC model for Neo4j (a popular graph database) that provides finer-grained access control by operating over resources.

CHAPTER 3: ON THE FEASIBILITY OF ATTRIBUTE-BASED ACCESS CONTROL POLICY MINING

Attribute-Based Access Control (ABAC) model has taken a considerable amount of time to get established, but now it is fast replacing other popular counterparts (e.g. RBAC, DAC) in industry, enterprise, and government applications. Due to the ongoing demand, switching to ABAC from an already employed access control system has acquired significant research interest. Automated or at least partially automated solutions can certainly reduce the amount of efforts needed and remove the errors caused by manual interventions but depend on the variety of input access control models. A specific offshoot of such problem, constructing equivalent ABAC policy from a given complete access control system (such as enumerated authorizations), and accompanying attribute data is called ABAC policy mining. In this chapter, we have identified a new problem called ABAC RuleSet Existence, in this context. The notion of ABAC RuleSet Infeasibility Correction has been introduced formally for the first time, along with algorithm analysis. In addition, promising future research directions have been discussed.

Significant portion of this chapter has been published at the following venues [13, 14].

- Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. 2019. On the Feasibility of Attribute-Based Access Control Policy Mining. In 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI). 245–252.
- Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. 2020. On the Feasibility of RBAC to ABAC Policy Mining: A Formal Analysis. In Secure Knowledge Management In Artificial Intelligence Era. Springer Singapore, Singapore, 147–163.

3.1 Motivation

To investigate the process of ABAC policy mining, let us consider two extreme cases.

1. All users have identical attribute values and likewise for all objects. Rules based on attribute values thereby cannot distinguish any two user, object pairs and can only give uniform authorization for all such pairs, which is hardly useful in practice.

2. At the other extreme assume, user identity and object identity are included as attribute values for users and objects respectively, where identity is globally unique. Every user, object pair is thereby distinguishable from every other pair, so authorization for each pair can be differentiated. In general, the inclusion of identity attributes will guarantee the existence of ABAC policy rules even if all other attributes are ignored.

We believe that identity attributes are antithetical to the spirit of ABAC and we disallow them, which makes the feasibility question germane. ABAC RuleSet Existence problem studies whether the intended ABAC policy generation is feasible or not under a set of criteria imposed, such as explicit unique ids are not allowed as attribute, equivalent ABAC policy is needed compared to the source access control system, etc. This problem has been investigated based on two types of source access control model, i) Enumerated Authorization System and ii) RBAC system. The notion of ABAC RuleSet Infeasibility Correction has been introduced and equivalent solutions are proposed as well. The rest of the chapter formulates ABAC RuleSet Existence Problem, associated terms, and related issues.

3.2 Preliminaries

In this section, some preliminaries of ABAC RuleSet Existence Problem will be noted and rest of the chapter will repeatedly use these definitions.

We consider access control systems that mediate access of users to objects. We specifically omit the user-subject distinction, e.g. as in [28]. Given that a user requests to perform an operation on an object, every access control system must define a checkAccess function to decide whether or not this operation should be permitted or denied.

Definition 1. checkAccess

$checkAccess$: $U \times O \times OP \rightarrow \{True, False\}$ where, U, O, and OP are finite sets of users, objects, and operations, respectively. A user $u \in U$ is allowed to perform operation $op \in OP$ on object $o \in O$ if and only if $checkAccess(u, o, op)$ is True.

In general, the checkAccess function changes with the system state. Since our focus is on a single state we omit explicit mention of the state. The specification of checkAccess, typically as a logical formula, depends upon the details of the underlying access control model.

3.2.1 Source Access Control System

As mentioned before, this chapter studies ABAC RuleSet Existence problem with two varieties of source access control system. A simple authorization system, where user-object-operation tuples are used directly to control access is as follows.

Definition 2. Enumerated Authorization System (EAS)
An EAS is a tuple $\langle$U, O, OP, AUTH, $checkAccess_{AUTH}\rangle$ where, U, O, and OP are finite sets of users, objects and operations, respectively. Here, $AUTH \subseteq U \times O \times OP$, is a specified authorization relation and $checkAccess_{AUTH}(u, o, op) \equiv (u, o, op) \in AUTH$.

For instance, user Paul can read object F if and only if (Paul, F, read)$\in$AUTH, whereby $checkAccess_{AUTH}$ is True. We require that the authorization state in the ABAC policy mining problem be given as an EAS. Note that however checkAccess is specified in an access control system, an equivalent AUTH relation can be computed for finite sets of users, objects and operations. So this is a reasonably general assumption.

The second variety of source access control system is RBAC system. The key component of RBAC system is role [21], an intermediary between user and permissions in the system. For example, all users assigned to a role "manager" may practice all permissions associated with that role. A complete RBAC system is defined as follows:

Definition 3. RBAC system
An RBAC system $\langle U, O, OP, Roles, RPA, RUA, RH, checkAccess_{RBAC}\rangle$ is a tuple where,

1. U, O, and OP are finite sets of users, objects, and operations, respectively.

2. $P = O \times OP$, is the set of all possible permissions in the system. A permission $p \in P$ is an object-operation pair where ops(p) and obj(p) denote the operation and object associated with p, respectively.

3. Roles is a finite set of role names.

4. The set of permissions directly assigned to a role $r \in Roles$ is given by RPA(r) where, $RPA\colon Roles \rightarrow 2^P$. The set of users directly assigned to a role $r \in Roles$ is given by RUA(r) where, $RUA\colon Roles \rightarrow 2^U$.

5. The role hierarchy relation is $RH \subseteq Roles \times Roles$ where RH must be acyclic. Here, $(r, r') \in RH$ denotes r is a senior role than r'.

6. Let reflexive transitive closure of RH be denoted by RH'. A role $r \in Roles$ acquires the set of permissions associated with all junior roles according to given hierarchy, and denoted by $authPerm(r) = \{p \in RPA(r') | (r, r') \in RH'\}$. A role $r \in Roles$ inherits all the users associated with seniors roles in hierarchy, and denoted by $authUser(r) = \{u \in RUA(r') | (r', r) \in RH'\}$.

7. Finally, $checkAccess_{RBAC}(u\colon U, o\colon O, op\colon OP) \equiv \exists r \in Roles.(u \in authUser(r) \wedge p \in authPerm(r) \wedge (o, op) = (obj(p), ops(p))$. In simple words, given a role $r \in Roles$, a user $u \in authUser(r)$ may practice all permissions $p \in authPerm(r)$.

Suppose the sets of users (U), objects (O), operations (OP) and roles (Roles) are $\{u1, u2, u3, u4, u5\}$, $\{o1, o2, o3\}$, $\{op1, op2\}$, and $\{r1, r2, r3, r4\}$, respectively. Given, $RH = \{(r1, r3)\}$, the user and permission assignment for each $role \in Roles$ is shown in Table 3.1. Here, user u1 can perform operation op1 on object o3 since $checkAccess_{RBAC}(u1, o3, op1)$ evaluates to True.

Table 3.1: RBAC system

Roles	RPA	RUA	authPerm	authUser
r1	{(o1, op1)}	{u1}	{(o1, op1), (o3, op1)}	{u1}
r2	{(o2, op2)}	{u3}	{(o2, op2)}	{u3}
r3	{(o3, op1)}	{u4, u5}	{(o3, op1)}	{u1, u4, u5}
r4	{(o1, op1), (o3, op1)}	{u2}	{(o1, op1), (o3, op1)}	{u2}

3.2.2 Target Access Control System

In this chapter, the target access control system is ABAC system. Before defining the ABAC RuleSet Existence problem, a complete specification of ABAC policy as well as the rule evaluation procedure is necessary.

Our ABAC model is adapted from [28] with two major deviations. Firstly, as mentioned above we omit the user-subject distinction. Secondly, attributes in [28] can be atomic-valued or set-valued. For example, the age attribute of a user is atomic valued. On the other hand a user's department attribute could be atomic valued if only one department is permitted or set-valued if a user can be in multiple departments. Note that set-valued attributes can be replaced by atomic-valued attributes by simply enumerating all combinations and assigning a symbol for each. For simplicity we assume all attributes are atomic valued.

In ABAC, authorization of whether a user can do an operation on an object is decided using the attributes value assignments of both user and object. (Many ABAC systems also include contextual attributes, which we ignore in this study.) The core of ABAC is a set of rules, which constitute the ABAC policy.

Definition 4. ABAC policy

An ABAC policy, POL_{ABAC} is a tuple, ⟨OP, UA, OA, RangeSet, RuleSet⟩ where,

- OP is a finite set of operations, and UA and OA are finite sets of user and object attribute function names respectively. We assume without loss of generality $UA \cap OA = \emptyset$.
- RangeSet = $\{(att, value) \mid att \in (UA \cup OA) \wedge value \in Range(att)\}$ where, Range(att) specifies a finite set of atomic values.

- RuleSet is a set of rules, where, for each operation op, RuleSet contains a single rule, $Rule_{op}$. Formally, RuleSet = $\{Rule_{op} \mid op \in OP\}$. Each $Rule_{op}$ is specified using the grammar below.

  $Rule_{op}$ ::= $Rule_{op} \vee Rule_{op} \mid (Atomicexp)$

  $Atomicexp$::=$Atomicuexp \wedge Atomicoexp | Atomicuexp | Atomicoexp \mid True \mid False$

  $Atomicuexp$ ::= $Atomicuexp \wedge Atomicuexp \mid uexp$

  $Atomicoexp$ ::= $Atomicoexp \wedge Atomicoexp \mid oexp$

  $uexp \in \{ua(u) = value \mid ua \in UA \wedge value \in Range(ua)\}$

  $oexp \in \{oa(o) = value \mid oa \in OA \wedge value \in Range(oa)\}$

Each $Rule_{op}$ is specified with user u and object o as formal parameters. The semantics of $Rule_{op}$, evaluated for an actual user a and object b is given in Definition 5. For example, suppose ABAC rule for read operation, denoted by $Rule_{read}$ is specified as (rank(u) = manager $\wedge$ type(o) = attendance log) $\vee$ (rank(u) = manager $\wedge$ type(o) = Annual report). Here any user in U with rank manager can read both types of objects, attendance log and annual report.

A complete ABAC system defines authorization based on ABAC policy as follows.

Definition 5. ABAC system

An ABAC system is a tuple, given by, $\langle$U, O, UAValue, OAValue, POL_{ABAC}, $checkAccess_{ABAC}\rangle$ where,

- U and O are finite sets of users and objects, respectively. OP, UA, OA, RangeSet and POL_{ABAC} are defined as in Definition 4.

- $UAValue = \{UAValue_{ua} | ua \in UA\}$ where the function $UAValue_{ua} \colon U \rightarrow Range(ua)$, such that $UAValue_{ua}(u)$ returns the value of attribute ua for user u. For convenience, we understand ua(u) to mean $UAValue_{ua}(u)$.

- $OAValue = \{OAValue_{oa} | oa \in OA\}$ where the function $OAValue_{oa} \colon O \rightarrow Range(oa)$, such that $OAValue_{oa}(o)$ returns the value of attribute oa for object o. For convenience, we understand oa(o) to mean $OAValue_{oa}(o)$.

- $checkAccess_{ABAC}$(a:U, b:O, op:OP) $\equiv Rule_{op}$(a:U, b:O) where $Rule_{op}$ is as stated in Definition 4. Given any user $a \in U$ along with attribute value assignments ua(a), where $ua \in UA$ and an object $b \in O$ along with attribute value assignment oa(b), where $oa \in OA$, the expression $Rule_{op}(a, b)$ is evaluated by substituting the values ua(a) for ua(u) and oa(b) for oa(o) in the $Rule_{op}$ expression. User a is permitted to do operation op on object b if and only if $Rule_{op}(a, b)$ evaluates to True.

We also define a **partially defined ABAC system** to be a tuple, $\langle$U, O, UAValue, OAValue, $POL_{ABAC-RuleSet}\rangle$ where $POL_{ABAC-RuleSet}$ is a tuple $\langle$OP, UA, OA, RangeSet$\rangle$ where OP, UA, OA, and RangeSet are as in Definition 4 and RuleSet is undefined.

Definition 6. Equivalency

An EAS, $\langle$U, O, OP, AUTH, $checkAccess_{AUTH}\rangle$, and an ABAC system, $\langle$U, O, UAValue, OAValue, POL_{ABAC}, $checkAccess_{ABAC}\rangle$ with identical U, O, and OP are said to be equivalent iff, $checkAccess_{AUTH}$(u, o, op) $\iff$ $checkAccess_{ABAC}$(u, o, op) for all $u \in U$, $o \in O$, and $op \in OP$.

3.3 ABAC RuleSet Existence Problem Definition

Based on the foregoing, ABAC RuleSet Existence problem where source access control system is EAS is defined as follows.

Definition 7. ABAC RuleSet Existence problem with EAS input

Given, an EAS $\langle$U, O, OP, AUTH, $checkAccess_{AUTH}\rangle$ and a partially defined ABAC system $\langle$U, O, UAValue, OAValue, $POL_{ABAC-RuleSet}\rangle$ where U, O and OP are identical to the given EAS, does there exist a RuleSet so that the resulting ABAC system is equivalent to the given EAS? Such a RuleSet, if it exists, is said to be a suitable RuleSet.

To illustrate the ABAC RuleSet Existence problem consider the example data in Table 3.2, where U = $\{u1, u2, u3, u4\}$, O = $\{o1, o2\}$, UA = $\{ua1, ua2\}$ and OA = $\{oa1\}$. Table 3.2(c) specifies the attribute ranges while Tables 3.2(a) and (b) gives attribute values for users and objects respectively. All of this specifies a partially defined ABAC system. Suppose we are given an EAS

Table 3.2: Example data set

(a) UAValue		
User	ua1	ua2
u1	F	C
u2	F	B
u3	F	C
u4	G	D

(b) OAValue	
Object	oa1
o1	F
o2	G

(c) Range	
ua1	$\{F, G\}$
ua2	$\{B, C, D\}$
oa1	$\{F, G\}$

with AUTH={(u1, o1, op)}. A suitable RuleSet cannot exist since users u1 and u3 cannot be distinguished based on their attribute values. However, if AUTH={(u1, o1, op),(u3,o1,op)} then a suitable $Rule_{op}$ is (ua1(u)=F $\wedge$ ua2(u)=C $\wedge$ oa1(o)=F).

Similarly, ABAC RuleSet Existence problem where source access control system is RBAC system is defined as follows.

Definition 8. ABAC RuleSet Existence problem with RBAC input

Given, an RBAC system and a partially defined ABAC system where U, O and OP are identical to the given RBAC system, does there exist a RuleSet so that the resulting ABAC system is equivalent to the given RBAC system? Such a RuleSet, if it exists, is said to be a suitable RuleSet.

To demonstrate the significance of the problem, let's consider the RBAC example in Table 3.1 and ABAC example in Table 3.3 with identical set of users, objects, and operations: does there exist a RuleSet so that the resulting ABAC system is equivalent to the given RBAC system? Note that it is always possible to generate equivalent ABAC system when explicit IDs are introduced for both user and object [45]. We strongly believe that the inclusion of such IDs is antithetical to the spirit of ABAC. Hence, we rule out the use of such IDs. For example, RBAC example in Table 3.1, user u1 can perform operation op1 on object o1 whereas user u3, a user with the same attribute value assignment as u1, is not allowed to do so. It is clearly evident that no suitable ABAC RuleSet can exist.

Table 3.3: ABAC example data

(a) UAValue	
User	uat1
u1	F
u2	F
u3	F
u4	G
u5	G

(b) OAValue	
Object	oat1
o1	F
o2	F
o3	G

(c) Range	
uat1	$\{F, G\}$
oat1	$\{F, G\}$

3.4 ABAC RuleSet Existence Problem with EAS input

In this section, solution to ABAC RuleSet Existence Problem with EAS input is discussed. In the case of infeasibility, solution is also provided.

3.4.1 Solution Approach

The essential concept to solve the ABAC RuleSet Existence problem is that the attribute name, value combinations induce a partition on the set of user, object pairs. We formalize this intuition as follows.

Definition 9. Binary relation R

Given, a partially defined ABAC system, $\langle$U, O, UAValue, OAValue, $POL_{ABAC-RuleSet}\rangle$, the binary relation R on set $UO = U \times O$ is defined as

$R \equiv \{((u1, o1), (u2, o2)) \mid (\forall ua \in UA.ua(u1) = ua(u2)) \wedge (\forall oa \in OA.oa(o1) = oa(o2))\}$

Lemma 1. *R is an equivalence relation.*

Proof: *Trivial by inspection.*

The resulting partitions induced by R on UO are formally referred to as follows.

Definition 10. Partition set P

Let $P = \{P_1, P_2, ..., P_n\}$ be the equivalence classes of R. Each $P_i \in P$ is called a partition element (or simply partition) and P is called the partition set. Each $P_i \in P$ is identified by a unique collection of (attribute name, value) pairs, given by $PV(P_i)$ where,

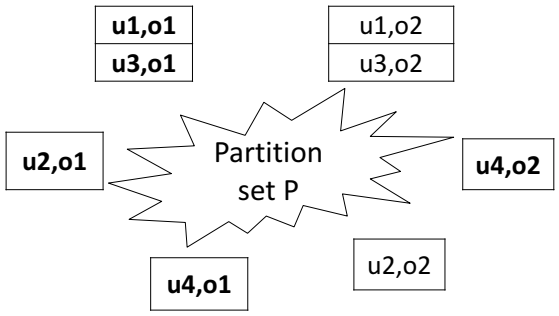

Figure 3.1: Partition set example.

$PV(P_i) \equiv$ For any $(u1, o1) \in P_i, (UV(u1) \cup OV(o1))$

UV(u:U) $\equiv \{(ua, value) | ua \in UA \wedge value = ua(u)\}$

OV(o:O) $\equiv \{(oa, value) | oa \in OA \wedge value = oa(o)\}$

For instance, using the example data in Table 3.2, UO = {(u1, o1), (u1, o2), (u2, o1), (u2, o2), (u3, o1), (u3, o2), (u4, o1), (u4, o2)}. The resulting partition set is shown in Fig. 3.1. The PV set for the partition containing (u1, o1) and (u3, o1) is $\{(ua1, F), (ua2, C), (oa1, F)\}$.

Finally, we introduce the following notion.

Definition 11. Conflict-free partition

Given an EAS, $\langle U, O, OP, AUTH, checkAccess_{AUTH} \rangle$ and partition set P, a $P_i \in P$ is conflict-free with respect to a specific $op \in OP$ iff the following statement is true,

$(\forall (u, o) \in P_i.(u, o, op) \in AUTH) \vee (\forall (u, o) \in P_i.(u, o, op) \notin AUTH)$

P_i has conflict with respect to $op \in OP$ otherwise. Partition set P is conflict-free iff for all $op \in OP$, every $P_i \in P$ is conflict-free with respect to op. P is called a conflicted partition set, otherwise.

An example of conflict-free partition set is presented in Fig. 3.1 where, user-object pairs in bold black belong to AUTH with respect to $OP = \{op\}$, while others are not. By inspection, all partitions in Fig. 3.1 are conflict-free with respect to given AUTH and $op \in OP$. Hence, resulting partition set is conflict-free. The concept of conflict-free partitions is used to solve ABAC RuleSet Existence problem as follows.

Theorem 1. *Given an ABAC RuleSet Existence problem instance, a suitable RuleSet exists iff P is conflict-free.*

Proof:

Only if *part is proved by contraposition. If P is not conflict-free, by definition, a conflict partition $P_i \in P$ with respect to a specific $op \in OP$, contains some $(u, o) \in P_i$ which are permitted in AUTH, while others are not. Since all $(u, o) \in P_i$ are represented by same $PV(P_i)$, these two logical parts of P_i cannot be separated using UAValue and OAValue. Thereby, $Rule_{op}$ generation is not possible. Note that this only if proof is independent of the actual policy language for $Rule_{op}$.*

If *part is proved by constructing a suitable RuleSet and showing that, if P is conflict-free then resulting ABAC system with generated RuleSet is equivalent to EAS. By definition, RuleSet contains a $Rule_{op}$, for each $op \in OP$. For a $op \in OP$, $Rule_{op}$ is given by:*

$$Rule_{op} = \bigvee_{P_i \times \{op\} \subseteq AUTH} (uexp(PV(P_i)) \wedge oexp(PV(P_i)))$$

$$uexp(PV(P_i)) = \bigwedge_{(ua,value) \in PV(P_i)} (ua(u) = value)$$

$$oexp(PV(P_i)) = \bigwedge_{(oa,value) \in PV(P_i)} (oa(o) = value)$$

To prove equivalency between the resulting ABAC system with RuleSet and EAS, it is necessary and sufficient to show that, for a specific op in OP, $(a, b, op) \in AUTH \iff Rule_{op}(a, b)$, for all $a \in U$, $b \in O$.

To prove the only if part: by inspection, each $(u, o) \in U \times O$ belongs to only one partition in P. Let $(a, b) \in P_i$. Since P is conflict-free, corresponding $P_i \times \{op\}$ must be a subset of AUTH. By construction, $Rule_{op}$ includes a conjunctive clause for every $P_i \times \{op\} \subseteq AUTH$, which evaluates to True for any user-object pair in P_i. Since $Rule_{op}$ is a disjunction of such conjunctive clauses, thereby, $Rule_{op}(a, b)$ evaluates to True. Hence, only if part is proved.

To prove if part: by inspection, if $Rule_{op}(a, b)$ evaluates to True, then there must be a conjunctive

clause of $Rule_{op}$, which is evaluated to True. By construction, each such conjunctive clause in $Rule_{op}$ is representing a specific $P_i \in P$ where, $P_i \times \{op\} \subseteq AUTH$. Since P is conflict-free, every user-object pair in corresponding P_i is permitted with respect to op, thus belongs to AUTH. Thereby, $(a, b, op) \in AUTH$, which proves if part.

Hence, given P is conflict-free, generated suitable RuleSet completes the ABAC system equivalent to given EAS.

Based on this result, a formal algorithm for ABAC RuleSet existence problem is presented in Algorithm 3.1.

Corollary 1. *Complexity of Algorithm 3.1 is $O(|OP| \times |U| \times |O|)$.*

Proof:

Given an equivalence relation R as in definition 9, the complexity of partition set P generation is $O(|U| \times |O|)$, considering partition creation, search, and insertion operations take constant time. For each $op \in OP$, checking whether P is conflict-free or not, requires $O(|OP| \times |U| \times |O|)$ as the maximum number of possible partition is $(|U| \times |O|)$. Hence, the overall complexity is $O(|OP| \times |U| \times |O|)$. Note that the size of any attribute range does not impact this complexity.

Simple rule generation will be illustrated with an example presented in Fig. 3.1. Since P is conflict-free with respect to given AUTH, using the rule construction procedure listed in Theorem 1, corresponding conjunctive clauses for each $P_i \in P$ where $P_i \times \{op\} \subseteq AUTH$ in Fig. 3.1 are $< ua1(u) = F \wedge ua2(u) = C \wedge oa1(o) = F >$, $< ua1(u) = F \wedge ua2(u) = B \wedge oa1(o) = F >$, $< ua1(u) = G \wedge ua2(u) = D \wedge oa1(o) = F >$, and $< ua1(u) = G \wedge ua2(u) = D \wedge oa1(o) = G >$. By construction, $Rule_{op}$ consists of disjunction of all the conjunctive clauses listed here. Now, any rule simplification approach can be used for further minimization.

3.4.2 Infeasibility Correction

We know from subection 3.4.1 that if the partition set is conflicted a suitable RuleSet cannot exist. There are at least two approaches to dealing with this situation in practice. One approach is to construct RuleSets that are only approximately equivalent to the given AUTH relation. Various

Algorithm 3.1 ABAC RuleSet Existence Algorithm

Require: EAS, Partially defined ABAC system where U, O, and OP are identical to EAS
Ensure: Partition set P and SUCCESS or FAILURE

Partition P := $\emptyset$
while $\exists (u, o) \in U \times O$ **do**
  if $\exists P_i \in P.PV(P_i) = (UV(u) \cup OV(o))$ **then**
    $P_i := P_i \cup \{\langle u, o \rangle\}$
  else
    Create new $P_i = \{\langle u, o \rangle\}$
    $P := P \cup P_i$
  $U \times O := U \times O - \{\langle u, o \rangle\}$
while $\exists op \in OP.\exists P_i \in P.!((P_i \times \{op\} \subseteq AUTH) \vee (P_i \times \{op\} \subseteq \overline{AUTH}))$ **do**
  return FAILURE
return SUCCESS

notions of approximation can be defined and their security implications analyzed. The second approach, which we study in this chapter, is to introduce additional attributes to reconcile the conflicted partitions. This leads us to introduce the following notion.

Definition 12. ABAC RuleSet Infeasibility Correction problem

Given, EAS $\langle U, O, OP, AUTH, checkAccess_{AUTH} \rangle$ and a partially defined ABAC system with unspecified RuleSet where, U, O, and OP are identical to the given EAS such that the partition set P is conflicted, the ABAC Ruleset Infeasibility Correction problem is to 1) add new attributes UA and/or OA, and 2) assign appropriate values to the added new attributes, so that it is possible to generate a suitable RuleSet.

Ideally, the newly added attributes should have semantic significance grounded in the underlying application domain, and should be assigned meaningful values appropriate to different users and objects. This will presumably require expert input from security architects, perhaps aided by artificial intelligence, machine learning and other automated techniques. Study of such approaches is beyond the scope of this chapter. Here we investigate a purely automated approach which introduces new "artifical" attributes with "artificial" values.

We note that conflict-free partitions that are authorized for access with respect to an operation op can have rules generated as explained in the proof of Theorem 1, ignoring consideration of any new attributes. However, the conflicted partitions need to be further refined by means of these new

attributes to remove the conflict. There can be many ways to do this. Clearly the minimum possible split of a conflicted partition is into two refined partitions, one with all user-object pair permitted for op while the second contains the denied ones. The maximum possible split is to put each tuple in the conflicted partition into its own refined partition. One possible approach to constructing appropriate refinements is described below.

Given ABAC RuleSet Infeasibility Correction instance, consider a $P_i \in P$ which is conflicted with respect to some $op \in OP$. Define the binary relation R_{Pi} on P_i as:

$R_{Pi} \equiv \{$((u1,o1), (u2,o2))$|\forall o \in O.\forall op \in OP.$((u1, o, op)$\in$ AUTH$\Leftrightarrow$(u2, o, op)$\in$ AUTH) $\wedge$ $\forall u \in U.\forall op \in OP.$((u, o1, op)$\in$AUTH$\Leftrightarrow$(u, o2, op)$\in$AUTH)$\}$

Lemma 2. *R_{Pi} is an equivalence relation.*

Proof: *Trivial by inspection.*

Hence, R_{Pi} induces a partition on P_i.

Definition 13. Partition set S_i

Let S_i be the partition on P_i induced by R_{Pi} and denoted by $S_i = \{S_{i1}, S_{i2}, ..., S_{im}\}$, where $1 \leq m \leq |P_i|$. Each $S_{ik} \in S_i$ is called a partition element (or shortly partition) and S_i is called partition set. By definition of R_{Pi}, each $S_{ik} \in S_i$ is represented by a collection of (attribute name, value) pairs, $PV(S_{ik})$ where,

$PV(S_{ik}) \equiv$ For any $(u1, o1) \in S_i, (UV(u1) \cup OV(o1))$

The concept of conflict-free partition from Section 3.4 is extended to S_i.

Definition 14. Conflict-free partition set S_i

Given an EAS, $\langle U, O, OP, AUTH, checkAccess_{AUTH}\rangle$ and partition set S_i stated in definition 13, a $S_{ik} \in S_i$ is conflict-free with respect to a specific $op \in OP$ and given $AUTH$ in EAS, iff the following statement is true:

$(\forall$(u,o) $\in S_{ik}$.(u,o,op) $\in AUTH) \vee (\forall$(u,o)$\in S_{ik}$.(u,o,op) $\notin$ AUTH)

S_{ik} has conflict with respect to $op \in OP$ otherwise. Partition set S_i is conflict-free with respect to given AUTH in EAS iff for all $op \in OP$, every $S_{ik} \in S_i$ is conflict-free with respect to op.

Table 3.4: Example data set

(a) UAValue	
User	uat1
u1	F
u2	F
u3	F
u4	G
u5	G

(b) OAValue	
Object	oat1
o1	F
o2	F
o3	F
o4	G

(c) Range of attributes	
uat1	$\{F, G\}$
oat1	$\{F, G\}$

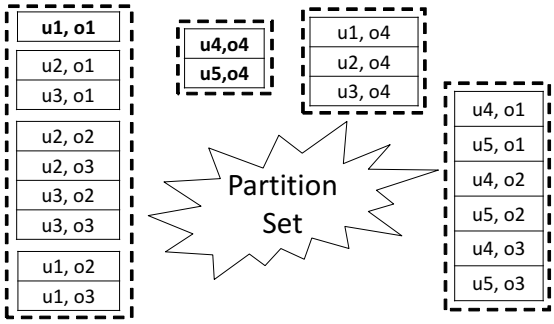

Figure 3.2: Refined partition set example.

Fig. 3.2 shows the resulting partitions for Table 3.4 where, bold black user-object pairs belong to AUTH with respect to OP = $\{op\}$ and rest are not. Here, U = $\{u1, u2, u3, u4, u5\}$, O = $\{o1, o2, o3, o4\}$, UA = $\{uat1\}$, OA = $\{oat1\}$, and Table 3.4 shows user attribute value assignment (UAValue), object attribute value assignment (OAValue), and range of attributes in (a), (b), and (c), respectively. To make visual comparison, the dotted rectangles in Fig. 3.2 shows partition set P for Table 3.4 as defined in Section 3.4. The leftmost conflicted parition is refined into four sub-partitions.

Lemma 3. *Given a conflict partition $P_i \in P$ with respect to $op \in OP$, the following holds:*

a. S_i is conflict-free

b. S_i refines P_i

c. For all $S_{ik} \in S_i$, PV(S_{ik}) is the same

Proof:

By inspection of definition S_i, it is conflict-free. S_i refines P_i because for each $S_{ik} \in S_i$, $S_{ik} \subseteq P_i$.

Since each $P_i \in P$ is identified by an unique $PV(P_i)$ and S_i does the refinement only, therefore, for all $S_{ik} \in S_i$, PV(S_{ik}) is same and (c) follows.

Definition 15. Given a partition $P_i \in P$, let $uList_i$ and $oList_i$ denote the sets of users and objects present in P_i. Let $uList_i$ be further partitioned as follows: any two users $u1, u2 \in ulist_i$ belong to same partition iff $\forall op \in OP. \forall o \in O. (u1, o, op) \in AUTH \iff (u2, o, op) \in AUTH$. Let this assumption split $uList_i$ into q partitions, denoted by $\{ul_{i1}, ..., ul_{iq}\}$. Similarly let $oList_i$ be partitioned as follows: any two objects $o1, o2 \in olist_i$ belong to same partition iff $\forall op \in OP. \forall u \in U. (u, o1, op) \in AUTH \iff (u, o2, op) \in AUTH$. Let this assumption split $oList_i$ into r partitions, denoted by $\{ol_{i1}, ..., ol_{ir}\}$.

Lemma 4. $S_i = \{ul_{i1}, ..., ul_{iq}\} \times \{ol_{i1}, ..., ol_{ir}\}$.

Proof: *Trivial by inspection of definitions.*

Definition 16. Introducing new user and object attributes

If P_i is a conflict partition, the following steps are proposed where, UND specifies "Unknown" status of an attribute value assignment.

1. $UA = UA \cup exU$ and $OA = OA \cup exO$ where, exU and exO are new user and object attributes, respectively. Initially, for all $u \in U$, $exU(u) := UND$ and for all $o \in O$, $exO(o) := UND$.

2. To ensure clarity, $PV_{new}(S_{ik} \in S_i)$ is introduced.

   $PV_{new}(S_{ik}) \equiv$ For any (u1, o1) $\in S_{ik}$, $(UV_{new}(u1) \cup OV_{new}$(o1))

   UV_{new}(u:U) $\equiv \{(ua, value) | ua \in (UA \cup exU) \wedge value = ua(u)\}$

   OV_{new}(o:O) $\equiv \{(oa, value) | oa \in (OA \cup exO) \wedge value = oa(o)\}$

   Here, Range(exU) and Range(exO) are sets of unique random values where $Range(exU) \cap Range(exO) = \emptyset$. The sets of random values are chosen so that new attribute and corresponding range can be added in an automated manner without human provided input.

3. Given a $P_i \in P$, algorithm 3.2 is used to assign appropriate values to the newly added attributes. Inside partitionCorrection, each of the q partitions$\in \{ul_{i1}, ..., ul_{iq}\}$ is assigned an

unique random value from Range(exU). Hence, every user in the same partition gets the same exU value. Similarly, each of the r partitions$\in \{ol_{i1}, ..., ol_{ir}\}$ is assigned an unique random value from Range(exO). Hence, every object in the same partition gets the same exO value.

Note: By Definition 15, $uList_i$ and $oList_i$ is further partitioned using universal quantifications on the sets U, O, and OP. Thereby, regardless of conflict partitions, once the exU and exO values are assigned by algorithm 3.2, they remain unchanged throughout the entire RuleSet generation process.

Lemma 5. *Based on Definition 22, for each $S_{ik} \in S_i$, $PV_{new}(S_{ik})$ is unique.*

Proof: *Follows trivially from Lemma 4 and Definition 22.*

For instance, given a conflict partition, P_i in Fig. 3.2 where only (u1, o1) belongs to AUTH with respect to op, it is refined into four new partitions. Initially, $uList_i$ is $\{u1, u2, u3\}$ and $oList_i$ is $\{o1, o2, o3\}$. According to Algorithm 3.2, $uList_i$ is further partitioned into $\{\{u1\}, \{u2, u3\}\}$. Similarly, $oList_i$ is further partitioned into $\{\{o1\}, \{o2, o3\}\}$. The resulting refined partitions has same PV, given by $\{(uat1, F), (oat1, F)\}$. According to Definition 22 and Algorithm 3.2, let exU value for $\{u1\}$ and $\{u2, u3\}$ be 1 and 2, respectively. Similarly, $\{o1\}$ and $\{o2, o3\}$ are assigned 3 and 4 for exO, respectively. Thereby, resulting unique PV_{new} value for the refined partitions are {(uat1, F), (oat1, F), (exU, 1), (exO, 3)}, {(uat1, F), (oat1, F), (exU, 1), (exO, 4)}, {(uat1, F), (oat1, F), (exU, 2), (exO, 3)}, and {(uat1, F), (oat1, F), (exU, 2), (exO, 4)}, respectively.

Theorem 2. *Given, an ABAC RuleSet Infeasibility Correction problem instance, it is always possible to find a suitable RuleSet such that the resulting ABAC system is equivalent to given EAS.*

Proof:

The theorem will be proved by construction. For a specific $op \in OP$, $Rule_{op}$ construction steps are as follows. It is assumed that, partition set P construction does not depend on exU and exO.

1. *Each conflict-free partition $P_i \in P$ is included in $Rule_{op}$ as conjunctive clause (same as Theorem 1) where, $P_i \times \{op\} \subseteq AUTH$.*

Algorithm 3.2 PartitionCorrection

Require: Conflict partition P_i and corresponding ABAC Ruleset Infeasibility Correction instance
Ensure: Refined partition set S_i where each $PV_{new}(S_{ik} \in S_i)$ is unique

uL := $\{ul_{i1}, ..., ul_{iq}\}$ //Def. 15
oL := $\{ol_{i1}, ..., ol_{ir}\}$ //Def. 15
if $\exists u \in uList_i.exU(u) = UND$ **then**
  while $\exists partu \in uL$ **do**
    uRandom := $v \in Range(exU)$ such that $\forall u \in U \setminus partu.exU(u) \neq v$
    For all $u1 \in partu$, $exU(u1) := uRandom$
    $uL := uL \setminus partu$
if $\exists o \in oList_i.exO(o) = UND$ **then**
  while $\exists parto \in oL$ **do**
    oRandom := $v \in Range(exO)$ such that $\forall o \in O \setminus parto.exO(o) \neq v$
    For all $o1 \in parto$, $exO(o1) := oRandom$
    $oL := oL \setminus parto$
    return S_i //$\{ul_{i1}, ..., ul_{iq}\} \times \{ol_{i1}, ..., ol_{ir}\}$

2. *Each conflict partition $P_i \in P$ is refined by Definitions 15 and 22, which generate conflict-free partitions only and ensure that for each such $S_{ik} \in S_i$, $PV_{new}(S_{ik})$ is unique. For each of the resulting partition $S_{ik} \in S_i$, a conjunctive clause is included in $Rule_{op}$ only if $S_{ik} \times \{op\} \subseteq AUTH$. For conflict partitions only, $Rule_{op}$ is given by:*

$$Rule_{op} = \bigvee_{P_i \in cp(P)} (uexp(PV_{new}(S_{ik})) \wedge oexp(PV_{new}(S_{ik})))$$

where cp(P) = $\{P_i \in P$ is a conflict partition$\}$, $S_{ik} \in partitionCorrection(P_i)$, and $S_{ik} \times \{op\} \subseteq AUTH$.

$$uexp(PV_{new}(S_{ik})) = \bigwedge_{(ua,value) \in PV_{new}(S_{ik}) \wedge ua \in UA \cup exU} (ua(u) = value)$$

$$oexp(PV_{new}(S_{ik})) = \bigwedge_{(oa,value) \in PV_{new}(S_{ik}) \wedge oa \in OA \cup exO} (oa(o) = value)$$

Here, $Rule_{op}$ is disjunction of all the conjunctive clauses generated in step 1 and 2. By definition, RuleSet contains a $Rule_{op}$, for each $op \in OP$. Hence, RuleSet can be generated. To prove equivalency between the resulting ABAC system with RuleSet and EAS, it is necessary and sufficient

to show that, for a op in OP, $(a, b, op) \in AUTH \iff Rule_{op}(a, b)$ *where* $a \in U$, $b \in O$.
To prove the only if part: by inspection, $(a, b) \in U \times O$ *belongs to only one partition, a* $P_i \in P$. *If* P_i *is conflict-free with respect to op then* $P_i \times \{op\}$ *must be a subset of AUTH. Hence, step 1 works. If* P_i *is a conflict partition, step 2 works. Let, (a,b)*$\in S_{ik}$ *where,* $S_{ik} \in S_i$. *Hence,* $S_{ik} \times \{op\}$ *must be a subset of AUTH. Since* $Rule_{op}$ *is disjunction of all the conjunctive clauses generated in step 1 and 2, thereby,* $Rule_{op}(a, b)$ *evaluates to true. Hence, only if part is proved.*
To prove if part: by inspection, if $Rule_{op}(a, b)$ *evaluates to true, then there must be a conjunctive clause of* $Rule_{op}$, *which is evaluated to true. By construction, each such conjunctive clause in* $Rule_{op}$ *is representing a specific partition where* $partition \times \{op\} \subseteq AUTH$. *Since each such partition is conflict-free, every user-object pair in corresponding partition is permitted with respect to op, thus belongs to AUTH. Thereby,* $(a, b, op) \in AUTH$, *which proves if part. Hence, it can be concluded that generated suitable RuleSet proposed by the steps above, completes the ABAC system, and equivalent to given EAS.*

Based on last example, two partitions {(u1, o1)}, and {(u4, o4), (u5, o4)} are included in $Rule_{op}$. Hence, $Rule_{op}$ is $(uat1(u) = F \wedge oat1(o) = F \wedge exU(u) = 1 \wedge exO(o) = 3) \vee (uat1(u) = G \wedge oat1(o) = G)$ and the RuleSet is $\{Rule_{op}\}$. In this example, both exU and exO are used for RuleSet Infeasibility Correction. If every user in U is represented by distinct user attribute value combination, exU is not required. The same condition holds for objects and exO.

Asymptotic complexity of ABAC RuleSet Infeasibility Correction is $O(|OP| \times (|U| \times |O|)^3)$. Given a partition set P with conflict, checking whether each $P_i \in P$ is conflict-free or not takes $O(|OP| \times (|U| \times |O|))$. If a $P_i \in P$ is in conflict with respect to a $op \in OP$, Algorithm 3.2 is called to refine P_i only. Inside Algorithm 3.2, corresponding list of users in P_i is futher partitioned by comparing each user-user pair, hence takes $O(|U|^2)$ complexity. Similarly, list of objects in P_i is partitioned; hence takes $O(|O|^2)$ complexity. Since a partition cannot have more than $|U|$ users and $|O|$ objects, while loops inside PartitionCorrection have upper bound $O(|U|)$ and $O(|O|)$, respectively. Hence, overall asymptotic complexity of PartitionCorrection algorithm is $O((|U| \times |O|)^2)$. Thereby, overall complexity is given by $O(|OP| \times (|U| \times |O|)^3)$.

3.5 ABAC RuleSet Existence problem with RBAC input

In this section, solution to ABAC RuleSet Existence Problem with RBAC input will be discussed. In the case of infeasibility, solution is also provided. More examples can be found in [14].

3.5.1 Solution Approach

Given an RBAC system, it is trivial to find an equivalent $AUTH$ relation, such that $(u, o, op) \in AUTH \Leftrightarrow checkAccess_{RBAC}(u, o, op)$. Since RBAC system to AUTH conversion takes $O(|U| \times |O|)$ complexity, the partition-based solution from section 3.4 can be reused in RBAC context by simply deriving the equivalent AUTH relation for the given RBAC system.

It is apparent that the binary relation R as in Def. 9 is an equivalence relation and thereby induces a partition P on UO induced by R as in Def. 10. The idea of conflict-free partition is defined in RBAC context as follows:

Definition 17. Conflict-free partition

Given $\langle U, O, OP, Roles, RPA, RUA, RH, checkAccess_{RBAC}\rangle$ as an RBAC system and partition set P where U, O and OP are identical, a $P_i \in P$ is conflict-free with respect to a specific $op \in OP$ iff the following statement is true:

$\forall(u, o) \in P_i.checkAccess_{RBAC}(u, o, op) = True \vee \forall(u, o) \in P_i.checkAccess_{RBAC}\ (u, o, op) = False$

P_i has conflict with respect to $op \in OP$ otherwise. Partition set P is conflict-free with respect to given RBAC system iff for each $op \in OP$, every $P_i \in P$ is conflict-free. P is called a conflict partition set, otherwise.

It is shown in section 3.4 that given an AUTH relation and partially defined ABAC system, a suitable RuleSet exists iff partition set P is conflict-free. The overall asymptotic complexity of ABAC RuleSet Existence problem with EAS input in section 3.4 is $O(|OP| \times (|U| \times |O|))$. The construction of AUTH relation by enumerating every possible user-object-operation tuple from an RBAC system takes $O(|U| \times |O|)$ time, thus overall asymptotic complexity of determining ABAC

RuleSet Existence in RBAC context remains the same as section 3.4, $O(|OP| \times (|U| \times |O|))$. By definition, suitable RuleSet (Theorem 1, section 3.4) consists of $|OP|$ rules, one for each $op \in OP$. Each conflict-free partition $P_i \in P$ is included in $Rule_{op}$ as a conjunctive clause where, $P_i \times \{op\} \subseteq AUTH$. For a specific $op \in OP$, $Rule_{op}$ (Theorem 1, section 3.4) construction steps are shown below:

$$Rule_{op} = \bigvee_{P_i \times \{op\} \subseteq AUTH} (uexp(PV(P_i)) \wedge oexp(PV(P_i)))$$

$$uexp(PV(P_i)) = \bigwedge_{(ua,value) \in PV(P_i)} (ua(u) = value)$$

$$oexp(PV(P_i)) = \bigwedge_{(oa,value) \in PV(P_i)} (oa(o) = value)$$

Note*: Related examples can be found in [14].

If partition set P is not conflict-free, no suitable RuleSet exists (section 3.4). Hence, in order to make the equivalent ABAC system generation always possible, one possible approach is to ensure that P is always conflict-free. There can be many possible ways to achieve this, either exact or approximate. In this study, ABAC RuleSet Infeasibility Correction problem in RBAC context is defined as follows.

Definition 18. ABAC RuleSet Infeasibility Correction problem

Given, RBAC system and partially defined ABAC system with unspecified RuleSet where U, O, and OP are identical to the given RBAC system, and a conflicted partition set P, ABAC Ruleset Infeasibility Correction problem is adding new attributes to 1) only UA or only OA or, both UA, OA, and 2) assign appropriate values to the new attributes, so that suitable RuleSet generation is always possible.

In the next section, an exact solution algorithm is presented for ABAC RuleSet Infeasibility Correction problem with the help of role-based attributes.

3.5.2 Infeasibility Correction

It is already established that if partition set P is conflict-free an equivalent ABAC system generation is always possible, since each $P_i \in P$ is uniquely identified by attribute values. Given a conflict partition set P, new role-based attributes are added and values are assigned accordingly so that each conflict partition in P is split into conflict-free fragments uniquely identified by attribute values. Thereby, equivalent RuleSet can be generated. Here, each conflict partition is processed separately to prevent unnecessary split of conflict-free partitions.

According to the construction in [28], an RBAC system can be configured to equivalent ABAC system even if no user, subject and object attributes are provided. The role membership information of an RBAC system can be utilized to generate appropriate attribute sets and value assignments. We adapt the construction in [28] to our user-object context as set-valued role membership attributes and omit the subject notion of [28].

Definition 19. Role-based user attribute

Given ⟨U, O, OP, Roles, RPA, RUA, RH, $checkAccess_{RBAC}$⟩ as RBAC system tuple, role-based user attribute is a set-valued attribute, $uroleAtt\colon U \rightarrow 2^{Roles}$. For a user $u \in U$, $uroleAtt(u)$ = $\{r \in Roles | u \in authUser(r)\}$.

Definition 20. Role-based object attribute

Given ⟨U, O, OP, Roles, RPA, RUA, RH, $checkAccess_{RBAC}$⟩ as RBAC system tuple, role-based object attribute for a $op \in OP$ is a set-valued attribute, denoted by $oroleAtt_{op}\colon O \rightarrow 2^{Roles}$. For an object $o \in O$, $oroleAtt_{op}(o)$= $\{r \in Roles | p \in authPerm(r) \wedge (o, op) = (obj(p), ops(p))\}$.

Although $uroleAtt$ is set-valued by definition, it is treated specially in this study: same as an atomic attribute. In order to generate uexp, "value" is as given in the definition 19 and to evaluate "uroleAtt(u) = value" in rule expression, "=" is considered as set equality operator. Similarly, each role-based object attribute w.r.t. a $op \in OP$ is treated specially as an atomic attribute. In order to generate oexp, "value" is as given in the definition 20 and to evaluate "$oroleAtt_{op}$(o) = value" in rule expression, "=" is considered as set equality operator.

Lemma 6. *Given an RBAC system, one user attribute as in definition 19 and $|OP|$ object attributes as in definition 20 (for each $op \in OP$) are sufficient to generate equivalent ABAC system.*

Proof:

Follows from the RBAC to ABAC configuration in [28]. Let the set of user attributes, UA = $uroleAtt$ and set of object attributes, OA = $\{oroleAtt_{op} | op \in OP\}$. The attribute value assignments of user and object attributes are as in definitions 19 and 20, respectively. To generate an equivalent ABAC system, each $P_i \in P$ must be identified by unique PV values as well as partition set P should be conflict-free (Theorem 1). It is trivial to show that both conditions are true, thereby, equivalent ABAC system generation is always possible.

This unique property of role membership in RBAC system makes it independent of supporting attribute data. It is a significant difference as compared to given authorization relation in subsection 3.4.2 where, a user and an object attributes are added to the attribute sets and unique random values are assigned to resolve infeasibility issue. The unique random value generation can be considered as an additional task whereas role membership attributes eliminate the need for such values and promotes self-sufficiency. Although Lemma 6 specifies the sufficiency of the role-based attributes to make an equivalent ABAC system generation, a more practical scenario is where supporting attribute data are provided. Therefore, the following definitions and proofs are presented to resolve ABAC Infeasibility Correction problem when supporting attribute data are provided; so that the resulting partition set becomes conflict-free where each partition element is uniquely identified by attribute values.

Definition 21. Binary relation $\mathbf{R_{Pi}}$ on $\mathbf{P_i} \in \mathbf{P}$

$R_{Pi} \equiv \{$((u1,o1), (u2,o2))$|\forall o \in O.\forall op \in OP.$((u1, o, op)$\in$ AUTH$\Leftrightarrow$(u2, o, op)$\in$ AUTH) $\wedge$ $\forall u \in U.\forall op \in OP.$((u, o1, op)$\in$AUTH$\Leftrightarrow$(u, o2, op)$\in$AUTH)$\}$

By inspection, R_{Pi} is an equivalence relation (Lemma 2). Let, R_{Pi} induces a partition on P_i, say $S_i = \{S_{i1}, S_{i2}, ..., S_{im}\}$, where $1 \leq m \leq |P_i|$. Each $S_{ik} \in S_i$ is called a partition element (or shortly partition) and S_i is called partition set. By definition, S_i further refines the partition P_i. Given a partition $P_i \in P$, let $uList_i$ and $oList_i$ denote the sets of users and objects present in

P_i. By inspection of definition of R, $P_i = uList_i \times oList_i$. Let $uList_i$ be further partitioned as follows: any two users $u1, u2 \in ulist_i$ belong to same partition iff $\forall op \in OP. \forall o \in O. (u1, o, op) \in AUTH \iff (u2, o, op) \in AUTH$. Let this assumption split $uList_i$ into q partitions, denoted by $\{ul_{i1}, ..., ul_{iq}\}$. Similarly let $oList_i$ be partitioned as follows: any two objects $o1, o2 \in olist_i$ belong to same partition iff $\forall op \in OP. \forall u \in U. (u, o1, op) \in AUTH \iff (u, o2, op) \in AUTH$. Let this assumption split $oList_i$ into r partitions, denoted by $\{ol_{i1}, ..., ol_{ir}\}$.

Lemma 7. S_i = $\{ul_{i1}, ..., ul_{iq}\} \times \{ol_{i1}, ..., ol_{ir}\}$ *and it is conflict-free.*

Proof: Trivial (Lemma 4).

Given a conflict partition $P_i \in P$, S_i has to be conflict-free and each $S_{ik} \in S_i$ should be identified uniquely by attribute values. The given set of attributes are not sufficient to serve this purpose unless there is some change in given attribute value assignments. The following definition adds the already defined role-based attributes to the given attribute set:

Definition 22. Add new role-based user and object attributes

Given ABAC RuleSet Infeasibility Correction instance, the following steps are proposed.

1. $UA_{new} = UA \cup uroleAtt$ and OA_{new} =OA $\cup \{oroleAtt_{op} | op \in OP\}$. Hence, total $1+|OP|$ attributes are added.

   Note: Initially, all new attributes are assigned UND which specifies "Unknown" attribute value assignment.

2. To ensure clarity, $PV_{new}(S_{ik} \in S_i)$ is introduced.

   $PV_{new}(S_{ik}) \equiv (UV_{new}(u1) \cup OV_{new}(o1))$ for any $(u1, o1) \in S_{ik}$ where

   UV_{new}(u:U) $\equiv \{(ua, value) | ua \in UA_{new} \wedge value = ua(u)\}$

   OV_{new}(o:O) $\equiv \{(oa, value) | oa \in OA_{new} \wedge value = oa(o)\}$

Lemma 8. *Given a conflict partition $P_i \in P$ w.r.t. a op $\in OP$, $PV_{new}(S_{ik})$ is unique.*

Proof:

By inspection of definition of R, for each $P_i \in P$, $PV(P_i)$ is unique. By definition, S_i further

refines the partition P_i. Hence, if it is proved that, given a conflict partition P_i w.r.t. a $op \in OP$, new user attribute can uniquely identify each element of $\{ul_{i1}, ..., ul_{iq}\}$ and similarly, $|OP|$ object attributes can do the same for $\{ol_{i1}, ..., ol_{ir}\}$, then $PV_{new}(S_{ik})$ is unique.

If $u1 \in ul_{im}$ and $u2 \in ul_{in}$ where $m \neq n$, let $uroleAtt(u1) = uroleAtt(u2)$. If $uroleAtt(u1) = uroleAtt(u2)$ then u1 and u2 cannot belong to two different partitions of $uList_i$ since it ensures uroleAtt(u1) and uroleAtt(u2) derives the exactly same set of permissions. Hence, $uroleAtt(u1) \neq uroleAtt(u2)$ proves. Thereby, each element of $\{ul_{i1}, ..., ul_{iq}\}$ is uniquely identified by uroleAtt value. However, given u3, u4$\in ul_{im}$, it is possible that $uroleAtt(u3) \neq uroleAtt(u4)$, although the resulting permissions are the same. By inspection, Algorithm 3.2 picks the minimum cardinality role set as role-based attribute value for every user in ul_{im}. Similarly, it can be proved that, If $o1 \in ol_{im}$ and $o2 \in ol_{in}$ where $m \neq n$, $\exists op \in Op.oroleAtt_{op}(o1) \neq oroleAtt_{op}(o2)$. Thereby, $PV_{new}(S_{ik})$ is unique.

Lemma 9. *Given $P_i = uList_i \times oList_i$ and $P_j = uList_j \times oList_j$, if $u1 \in uList_i$ and $u1 \in uList_j$, then $uList_i = uList_j$.*

Proof:

Follows from definition of R, it is trivial. Similarly, it can be proved that, if $o1 \in oList_i$ and $o1 \in oList_j$, then $oList_i = oList_j$.

Note: In Algorithm 3.2, Lemma 9 is used to prevent repeated role-based attribute value assignment of users and objects. Based on the foregoing, the following theorem states and proves the solution of ABAC RuleSet Infeasibility Correction problem.

Theorem 3. *Given an ABAC RuleSet Infeasibility Correction problem instance as in Def. 18, it is always possible to find a suitable RuleSet such that the resulting ABAC system is equivalent to given RBAC system (adapted from Theorem 2).*

Proof:

Given an RBAC system, equivalent AUTH relation is generated first. Given a $op \in OP$, the $Rule_{op}$ construction procedure is described below. Here, partition set P construction entirely depend on the given attribute set only (no role-based attributes).

1. Each conflict-free partition $P_i \in P$ is included in $Rule_{op}$ as conjunctive clause where, $P_i \times \{op\} \subseteq AUTH$. For a $op \in OP$, such $Rule_{op}$ is defined in Section 3.5.

2. After applying definition 22, each conflict partition $P_i \in P$ is further refined by Algorithm 3.2. By using Lemma 8, $\forall S_{ik} \in S_i$, $PV_{new}(S_{ik})$ is unique where each $S_{ik} \in S_i$ is conflict-free. A conjunctive clause is included in $Rule_{op}$ only if $S_{ik} \times \{op\} \subseteq AUTH$ where $S_{ik} \in S_i$. The following shows $Rule_{op}$ construction procedure for conflict partitions in P only:

$$Rule_{op} = \bigvee_{P_i \in CFP(P)} (uexp(PV_{new}(S_{ik})) \wedge oexp(PV_{new}(S_{ik})))$$

where CFP(P) consists of all conflict partitions in P with respect to $op \in OP$, $S_{ik} \in confRefine(P_i)$, and $S_{ik} \times \{op\} \subseteq AUTH$.

$$uexp(PV_{new}(S_{ik})) = \bigwedge_{(ua,value) \in PV_{new}(S_{ik})} (ua(u) = value)$$

$$oexp(PV_{new}(S_{ik})) = \bigwedge_{(oa,value) \in PV_{new}(S_{ik})} (oa(o) = value)$$

Here, $Rule_{op}$ is the disjunction of all the conjunctive clauses generated in step 1 and 2. By definition, RuleSet consists of total $|OP|$ rules, one for each $op \in OP$. Hence, a RuleSet can be constructed. To prove equivalency between the resulting ABAC system with constructed RuleSet and RBAC system, it is necessary and sufficient to show that, for a op in OP, $checkAccess_{RBAC}$(c,d,op) = True $\iff$ $Rule_{op}(c, d)$ where $c \in U$, $d \in O$ which implies $(c, d, op) \in AUTH \iff Rule_{op}(c, d)$.

The proof is divided into two parts: (i) only if and (ii) if. To prove (i): by inspection of partition and related definitions, $(c, d) \in U \times O$ belongs to only one partition in P. Let, $(c, d) \in P_i$ where $P_i \in P$. If P_i is conflict-free with respect to op then $\forall (u, o) \in P_i.(u, o, op) \in AUTH$ holds (step 1 in $Rule_{op}$ generation). If P_i is a conflict partition then step 2 is followed. Let, (c,d)$\in S_{ik}$ where $S_{ik} \in S_i$. Hence $\forall (u, o) \in S_{ik}.(u, o, op) \in AUTH$ holds. As a result, S_{ik} is included in $Rule_{op}$ as conjunctive clause (as per step 2 in $Rule_{op}$ construction procedure). Since $Rule_{op}$ consists of

disjunction of all the conjunctive clauses generated in step 1 and 2, $Rule_{op}(c, d)$ evaluates to true and (i) is proved.

The part (ii) of the proof: by inspection of $Rule_{op}$ construction stated above, if $Rule_{op}(c, d)$ evaluates to true then there exists a conjunctive clause of $Rule_{op}$ which turned into true. By $Rule_{op}$ construction procedure, each such conjunctive clause in $Rule_{op}$ is presenting a particular partition where for all $(u, o) \in partition.(u, o, op) \in AUTH$. Thereby, the statement $(c, d, op) \in AUTH$ is true and (ii) is proved. Hence, the constructed RuleSet completes the ABAC system, and equivalent to given RBAC system (proved by construction).

One notable optimization at this point is: role-based attributes should be used only when they are needed. For example, if each user in the given user set is represented by a unique user attribute value assignment then there is no need to introduce a role-based user attribute, even if the partition set is conflicted. The same strategy can be applied for role-based object attribute: if each object is represented by unique attribute value assignment, role-based object attributes are unnecessary even if partition set is conflicted. If both of the cases do not hold, still role-based attributes can be removed while generating a conjunctive clause for a particular conflicted partition. For a conflict partition $P_i \in P$ where $P_i = uList_i \times oList_i$, if $|uList_i| = 1$ then role-based user attribute can be avoided while generating conjunctive clauses for P_i. Similarly, role-based object attribute can be ignored when $|oList_i| = 1$.

The asymptotic complexity of ABAC RuleSet Infeasibility Correction in RBAC context is given by $O(|OP| \times (|U| \times |O|)^3)$, same as in in EAS context.

3.6 Unrepresented partitions

Given range of attributes and a specific set of attribute value assignment to users and objects, it is quite possible that some attribute value combinations will not show up while generating partition set. We call these partitions as “unrepresented", since the range of attributes clearly allows presence of those, but due to the peculiarity in the given user and object attribute value assignment, these partitions remain empty. For instance, all possible attribute value combinations (each

one represents a possible partition) for Table 3.2 data is given by {FCF, FBF, GDF, GDG, FCG, FBG, FDF, FDG, GBF, GCF, GBG, GCG}, considering an order of $< ua1, ua2, oa1 >$. Only first six combinations of this set are present, while the remaining six are unrepresented according to the given data. The ABAC policy mining approaches in [42, 47] ignore unrepresented partitions, whereby the generated rules may or may not authorize these attribute value combinations to have access. If these unrepresented partitions get populated in future, this may lead to unexpected checkAccess decisions.

To have an insight using the same data set for ABAC policy mining where authorizations are presented in Fig. 3.1, [47] derives two rules without user and object id, $<true$, $true$, $\{op\}$, $\{ua1$=$oa1\}>$ and $<ua1$=$\{G\}$, $true$, $\{op\}$, $\emptyset>$. Since [47] works fine for all possible user-object pair, first four of the attribute value combinations are allowed, while FCG and FBG are denied. Amongst unrepresented attribute value combinations, $\{GCG, GBG, FDF\}$ satisfies the first clause of the rule, and $\{GBF, GCF, GBG, GCG\}$ are accepted by the second clause. Only $\{FDG\}$ gets rejection according to the generated rules! The question of unrepresented attribute combinations is treated as a rule simplification concern, rather than a security concern in [47]. Another approach in [42] generates two rules, $< (oa1 = F) >$ and $< (ua2 = D) >$ and works fine for given authorizations. In case of unrepresented partition, $\{FDF, FDG, GBF, GCF\}$ gets acceptance, while $\{GBG, GCG\}$ is denied. This mixed response shows "don't care" for unrepresented attribute value combinations again. The security architect should be at least aware of these unrepresented combinations. He might decide to don't care, or take suitable action based on his expertise. Both of our approaches defined in the previous sections deny access to unrepresented partitions which is a conservative and safe security posture.

3.7 Use Cases and Implementation

More use cases of ABAC RuleSet Existence Problem can be found in [13, 14]. For example, given a complete RBAC system, the role membership information of an RBAC system can be utilized to generate appropriate attribute sets and value assignments. According to Lemma 6, the set of

Table 3.5: Role-based attribute values for RBAC system in Table 3.1

Objects	$oroleAtt_{op1}$	$oroleAtt_{op2}$
o1	{r1, r4}	{}
o2	{}	{r2}
o3	{r1, r3, r4}	{}

Users	$uroleAtt$
u1	{r1, r3}
u2	{r4}
u3	{r2}
u4	{r3}
u5	{r3}

role-based attributes and corresponding value assignment of RBAC system in Table 3.1 are shown in Table 3.5.

The implementation part in feasibility of ABAC policy mining is a simple module which resolves ABAC RuleSet Existence Problem with EAS input and generates the final ABAC rule. In the event of infeasibility, the implementation adds additional user and object attributes as mentioned earlier, thus ABAC rule generation is always feasible. A simple Java implementation of the aforementioned with sample randomly generated cases is available upon request.

CHAPTER 4: FORMAL ANALYSIS OF REBAC POLICY MINING FEASIBILITY

Relationship-Based Access Control (ReBAC) expresses authorization in terms of various direct and indirect relationships amongst entities, most commonly between users. The need for ReBAC policy mining arises when an existing access control system is reformulated into ReBAC. This chapter considers the feasibility of ReBAC policy mining in context of user to user authorization, such as arises in various social and business contexts. In accordance with the policy mining literature, we assume that complete data is provided regarding user to user authorizations for a given user set, along with complete relationship data amongst these users comprising a labeled relationship graph. A ReBAC policy language is also specified. ReBAC policy mining seeks to formulate a ReBAC policy with the given policy language and relationship graph, which is exactly equivalent to the given authorizations. ReBAC policy mining feasibility problem asks whether such a policy exists and if so to provide the policy. We investigate this problem in context of different ReBAC policy languages which differ in the relationships, inverse relationships and non-relationships that can be used to build the policy. We develop a feasibility detection algorithm and analyze its complexity. We show that our policy languages are progressively more expressive as we introduce additional capability. In case of infeasibility, various solution approaches are discussed.

This chapter has been published at the following venue [11].

- Shuvra Chakraborty and Ravi Sandhu, Formal Analysis of ReBAC Policy Mining Feasibility. In Proceedings of the 11th ACM Conference on Data and Application Security and Privacy (CODASPY), April 26-28, 2021, Virtual Event.

4.1 Motivation

ReBAC policy mining problem seeks to automate the process of obtaining an ReBAC policy when a complete access control system along with supporting relationship data is given. ReBAC policy mining algorithms offer promising advancement in automating policy generation, whereas manual

effort requires more time, and could be error-prone.

ReBAC policy mining approaches such as [5, 7–10] permit use of the unique identity (id) of entities (e.g., users and resources) in the generated ReBAC policies. Hence, ReBAC policy mining is always feasible. We believe that use of such ids is contrary to the core ReBAC spirit. Thereby, determining feasibility becomes a significant question in mining ReBAC policies. In case of infeasibility, we propose various solutions as an alternative to using ids in ReBAC policy generation.

In this chapter, we investigate the ReBAC policy mining approach from a novel perspective. We study the feasibility of the ReBAC policy mining process, in context of various ReBAC rule structures.

4.2 Preliminaries

In this section, some preliminaries of ReBAC RuleSet Existence Problem will be noted and rest of the chapter will repeatedly use these definitions.

A user is an entity who performs operations (also called actions). An operation is an act performed by a user on another user. A user can be an initiator or a target of an operation. The finite (but unbounded) set of current users is denoted as U. The finite set of operations is denoted by OP, where each operation in OP is independently authorized.

Given that a user requests to perform an operation on another user, every access control system must define a checkAccess function to decide whether or not this operation is permitted or denied. The specification of checkAccess, typically as a logical formula, depends upon the details of the underlying access control model.

Definition 23. checkAccess

$checkAccess\colon U \times U \times OP \rightarrow \{True, False\}$ where U and OP are finite sets of users and operations respectively. A user $u \in U$ is allowed to perform operation $op \in OP$ on a user $v \in U$ iff $checkAccess(u, v, op)$ is True.

Without loss of generality, we assume OP is the singleton set $\{op\}$, since each operation is independently authorized. For simplicity, OP is thereby omitted from further definitions. For a specific

access control model M we write $checkAccess_M(u, v)$.

An access request is a tuple $\langle u, v\rangle$, where $u, v \in U$ and $u \neq v$, which specifies user u has requested to perform operation op on user v.

4.2.1 Source Access Control System

This chapter studies ReBAC RuleSet Existence problem with a single type of source access control system. A simple authorization system, where user to user tuples are used directly to control access authorization is as follows.

Definition 24. Enumerated Authorization System (EAS)

An EAS is a tuple $\langle U, AUTH, checkAccess_{EAS}\rangle$ where, U is the finite set of users, $AUTH \subseteq U \times U$, is the authorization relation where $\forall(u, v) \in AUTH.u \neq v$, and $checkAccess_{EAS}(u, v) \equiv (u, v) \in AUTH$.

Given the set of users $U = \{Alice, Bob, Cathy\}$ and $AUTH = \{(Alice, Bob), (Bob, Cathy)\}$, an access request $\langle Alice, Bob\rangle$ is granted whereas $\langle Alice, Cathy\rangle$ is denied. $AUTH$ is essentially an access matrix.

It is a trivial task to convert an access control system to EAS, therefore, various type source access control system can be accommodated.

4.2.2 Target Access Control System

In this chapter, the target access control system is ReBAC system. Before defining the ReBAC RuleSet Existence problem, a complete specification of ReBAC policy as well as the rule evaluation procedure is necessary.

Relationships are represented as a directed labeled graph.

Definition 25. Relationship Graph (RG)

The Relationship Graph RG = (V, E, Σ) of a system is a directed labeled graph where,
i) V is the set of vertices in RG, representing the current set of users, ii) $E \subseteq V \times V \times \Sigma$ is a finite

set of labeled directed edges where Σ is a finite set of relation type specifiers.

An edge $(u, v, \sigma) \in E$, $u \neq v$, represents the relation $\sigma \in \Sigma$ from user $u \in V$ to $v \in V$ in RG where σ is the edge label.

For example, consider the RG in Fig. 4.1 where V = { Alice, Bob, Cathy }, E = {(Alice, Bob, F)}, and $\Sigma = \{F\}$ (F represent the friend relation). Then Alice is a friend of Bob, but not vice versa, whereas Cathy is a completely isolated user.

Direct relationships are represented as edges in RG, while indirect relationships are represented as paths. For our purpose, it is convenient to define path in two steps as follows.

Definition 26. Linked Sequence of Vertices

Given RG = (V, E, Σ) and a vertex pair $(u, v) \in V \times V$ where $u \neq v$, a (simple) linked sequence of vertices is a set of triples where the terminating (i.e., second) vertex of each triple is same as the starting (i.e., first) vertex of the next triple given by $\langle (u, v_i, \sigma_w),\ (v_i, v_j, \sigma_x),\ ...,\ (v_k, v_l, \sigma_y), (v_l, v, \sigma_z) \rangle$, where $u, v_i, v_j, ..., v_k, v_l, v \in V$, and $\sigma_w, \sigma_x, ..., \sigma_y, \sigma_z \in \Sigma$, such that once a vertex v_i occurs as a start vertex it cannot be the terminating vertex in subsequent triples.

Definition 27. Path in Relationship Graph

A (simple) linked sequence of vertices is a (simple) path from u to v if each triple belongs to E in RG, i.e., it is an edge. The path label of a path is $\sigma_w \sigma_x .....\sigma_y \sigma_z$. Its length is the number of triples, or equivalently the number of symbols in the path label.

Since we only consider simple paths in this study we will often drop the simple qualifier. It should be noted that Def. 26 and 27 would traditionally be merged to define a path, but separating them makes it convenient to define path variations later in Def. 31. For convenience, given a path p in RG we understand pathLabel(p) to denote the path label of path p.

A crucial component of ReBAC is a set of rules called the ReBAC policy, formally defined as follows:

Definition 28. ReBAC Policy

A ReBAC policy, POL_{ReBAC} is a tuple, given by $\langle \Sigma, RuleSet \rangle$ where:

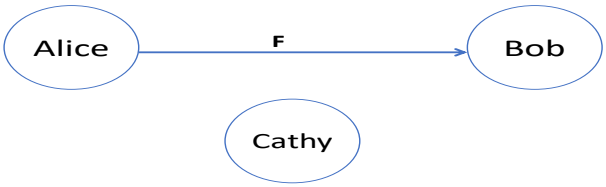

Figure 4.1: Example Relationship Graph

- Σ denotes the finite set of relation type specifiers in the system.
- RuleSet is a set of rules where, for each operation $op \in OP$, RuleSet contains the single rule $Rule_{op}$. Each $Rule_{op}$ is specified using the grammar below.

  $Rule_{op} ::= Rule_{op} \vee Rule_{op} \mid pathRuleExpr$

  $pathRuleExpr ::= pathRuleExpr \wedge pathRuleExpr \mid pathLabelExpr$

  $pathLabelExpr ::= pathLabelExpr.pathLabelExpr \mid edgeLabel$

  $edgeLabel ::= \sigma, \sigma \in \Sigma$

Here "." is the concatenation operator. As stated earlier, it suffices to consider OP to be a singleton, so RuleSet consists of a single rule. The $Rule_{op}$ expression consists of disjunction of pathRuleExpr, where each pathRuleExpr consists of conjunction of pathLabelExpr. The pathLabelExpr is a concatenated string of relationship type specifiers. The $Rule_{op}$ evaluation procedure is described in Def. 29.

This leads to the following definition of a ReBAC system.

Definition 29. ReBAC System

A ReBAC system is a tuple, $\langle RG, POL_{ReBAC}, checkAccess_{ReBAC} \rangle$ where $checkAccess_{ReBAC}$*(a:V, b:V)* is evaluated as follows: (i) for each pathLabelExpr in $Rule_{op}$ substitute True if there exists a simple path p from a to b in RG with path label pathLabelExpr, otherwise substitute False, (ii) evaluate the resulting boolean expression.

For example, consider Fig. 4.1 with $Rule_{op} = F$. Given an access request $\langle$Alice, Bob$\rangle$, there is a

simple path from Alice to Bob with path label F so True is substituted for F and $Rule_{op}$ evaluates to True whereby the access request is granted.

4.3 ReBAC RuleSet Existence Problem

This section develops the formal definition of the ReBAC RuleSet Existence Problem (RREP). As we are going to investigate variations of RREP later in this study, we call the the core RREP problem defined in this section as RREP-0.

The definitions provided above bring us to definition of the central problem addressed in this study.

Definition 30. ReBAC RuleSet Existence Problem (RREP-0)
Given an EAS = $\langle$U,AUTH,$checkAccess_{EAS}\rangle$ and RG = (V,E,Σ) with V=U, does there exist a RuleSet as in Def. 28 so that the resulting ReBAC system satisfies:

$$(\forall u, v \in U)[checkAccess_{ReBAC}(u, v) \Leftrightarrow checkAccess_{EAS}(u, v)]$$

Such a RuleSet, if it exists, is said to be a suitable RuleSet, otherwise the problem is said to be infeasible.

For example, for the RG in Fig. 4.1, a suitable RuleSet exists only if the given AUTH = $\{(Alice, Bob)\}$, with $Rule_{op} = F$. Any other AUTH relation will not have a suitable ReBAC RuleSet.

RuleSet Generality

A natural question to investigate at this point is the generality of our ReBAC RuleSet structure. We consider three criteria in this regard: variety of entities available, expressiveness of policy language and relationship depth. In this study, we limit our scope to user to user relationships only which is a common case in OSNs. More generally, ReBAC policy mining may incorporate multiple entity types. For example, [8] uses the familiar class-object concept in their ReBAC rules, where each class represents a a particular entity type and an object is an instance of a class. In comparison with [8], we deal with a single class User. We discuss expressiveness in Section 4.4 and will

show by examples that our rule structure is not the most general one. Given a vertex pair (a,b) in RG and simple path p from a to b, the relationship depth is the length of the path. With finite RG, the relationship depth is inherently limited by the maximum simple path length between any vertex pair in RG. In [8], relationship depth is provided as algorithm input. While in our work relationship depth is not provided as input, a slight modification would accommodate this. Specifically, adding a constraint in line 5 of Algorithm 4.1 to limit the maximum simple path length to a provided value. Other constraints could be similarly enforced. For example, a constraint that limits the number of path labels used in the conjunctive term generation in line 16-17 of Algorithm 4.1 to a given numeric value.

Algorithm 4.1 ReBAC RuleSet Existence Problem-0 Algorithm

Input: An EAS $\langle$ U,AUTH,$checkAccess_{EAS}\rangle$ and a RG = (V,E,Σ) where V=U

Output: Feasible/infeasible status. If feasible $\rightarrow$ generate ReBAC rule, return "infeasible" and set of infeasible authorization tuples otherwise.

1: $Rule_{op} := NULL$
2: failedAuthList := $\emptyset$
3: AUTHset := AUTH //copying AUTH
4: **while** $\exists(a,b) \in AUTHset$ **do**
5: $SP(a,b)$:= FindAllSimplePath(a,b, RG) // Algo. 4.2
6: **if** $SP(a,b) = \emptyset$ **then**
7: $failedAuthList := failedAuthList \cup \{(a,b)\}$ //Not Feasible for (a,b) tuple
8: $AUTHset \setminus := \{(a,b)\}$ and **Continue**
9: $PATHLABEL(a.b) := \{pathLabel(p) | p \in SP(a,b)\}$
10: **for** each $pl \in PATHLABEL(a.b)$ **do**
11: $SAT_{ab}(pl)$ = $\{(c,d) \in V \times V |$ there exists a simple path s from c to d in RG, c$\neq$d, (c,d)$\notin$AUTH, pl=pathLabel(s)$\}$
12: $Q_{ab} := \bigcap_{pl \in PATHLABEL(a.b)} SAT_{ab}(pl)$
13: **if** $Q_{ab} \neq \emptyset$ **then**
14: $failedAuthList := failedAuthList \cup \{(a,b)\}$ //Not Feasible for (a,b) tuple
15: $AUTHset \setminus := \{(a,b)\}$ and **Continue**
16: **if** $Rule_{op}$ is NULL **then** $Rule_{op} := \bigwedge_{pl \in PATHLABEL(a.b)} pl$ **else** $Rule_{op} := Rule_{op} \vee \bigwedge_{pl \in PATHLABEL(a.b)} pl$
17: $AUTHset \setminus := \{(a,b)\}$
18: **if** failedAuthList is $\emptyset$ **then return** ("feasible", $Rule_{op}$) **else return** ("infeasible", $Rule_{op}$, failedAuthList)

Algorithm 4.2 FindAllSimplePath

Input: Vertex source, vertex dest, $RG = (V, E, \Sigma)$

Output: Set of all simple paths from source to dest in RG

1: //visitVertex is a map where visitVertex[$u \in V$] = white means "not visited", visitVertex[$u \in V$] = grey means "visited but not finished yet"
2: //visitEdge is a map where visitEdge[$e \in E$]=white means "not visited", visitEdge[$e \in E$]=grey means "visited but not finished yet"
3: **for** $u \in V$ **do**
4: visitVertex[u]:=white
5: **for** $e \in E$ **do**
6: visitEdge[e]:=white
7: PS := $\emptyset$
8: Modified-DFS-Visit(source, dest, RG, PS, $\langle\rangle$) //assuming visitVertex and visitEdge are globally defined
9: **return** PS

Algorithm 4.3 Modified-DFS-Visit

Input: vertex src, vertex dest, $RG(V, E, \Sigma)$, PS, tempPath

Output: Path generation from src to dest in RG

1: **if** src == dest **then**
2: $PS\cup := tempPath$
3: **return**
4: visitVertex[src]:=grey
5: **for** each edge $e \in E$, where e=(x,y,σ) and x=src **do**
6: **if** visitEdge[e] == white and visitVertex[y] == white **then**
7: Modified-DFS-Vist(y,dest,RG,PS,appendSeq(tempPath,e)) //appendSeq() is a trivial function which appends edge e to the path sequence, tempPath and returns the new ordered path sequence
8: visitVertex[src]:=white
9: **for** each edge $e \in E$, where e=(x,y,σ) and x=src **do**
10: visitEdge[e] := white
11: **return**

4.3.1 Feasibility Detection Algorithm

In this subsection, the feasibility detection Algorithm 4.1 for RREP-0 is presented along with proofs and complexity analysis. The algorithm iterates through each tuple $(a, b) \in AUTH$, and either finds a rule that is correct for (a, b) or deems the tuple to be infeasible and records it in failedAuthList. In each iteration it computes all possible simple paths from a to b to find whether the resulting collection of pathLabels is collectively satisfied by any unauthorized tuple (i.e., a tuple not in $AUTH$). The function FindAllSimplePath, which is able to find all simple paths between any vertex pair in RG, is described briefly in Algorithm 4.2 and 4.3. For completeness, the algorithm used for all possible path generation in Algorithm 4.1, called FindAllSimplePath has been included. Given a RG and a vertex pair (source,dest), algorithm FindAllSimplePath returns the set of all possible simple paths from source to dest in RG. It is basically a modified form of core Depth-First-Search from vertex source to vertex dest in RG.

If (a,b) is disconnected in RG then it is infeasible. Otherwise, all possible pathLabels for (a,b) are generated in line 9 as in Def. 27. If there exists any unauthorized vertex pair which satisfies all possible pathLabels from a to b, (a,b) is infeasible as in line 14. The $Rule_{op}$ is updated otherwise and (a,b) is removed from further consideration, as shown in line 16-17.

At the end, if rule generation is not feasible for any particular tuple in AUTH, i.e., failedAuthList is not empty, the algorithm returns an infeasible result along with all infeasible tuples in failedAuthList. Another alternative is to abort at the point where first infeasible tuple is encountered if failedAuthList is not available. If rule generation is feasible for every tuple $(a, b) \in AUTH$, $Rule_{op}$ is generated and feasible status is returned.

Theorem 4. *The overall complexity of RREP-0 feasibility detection Algorithm 4.1 is* $O(|V|^4 \times (|E|!))$.

Proof:

In order to compute Algorithm 4.1 complexity, Algorithm 4.2 and 4.3 are needed to be considered first. Algorithm 4.2 finds the set all possible simple paths between a vertex pair in RG using a

variant of DFS in Algorithm 4.3. Since it considers only simple path, the overall complexity of Algorithm 4.2 is $O(|E|!)$, considering $|V| \leq |E|$. Therefore, the complexity of line 5 and 9 in Algorithm 4.1 is $O(|E|!)$. In line 10-11, the SAT_{ab} function computation takes overall $O(|V|^2 \times (|E|!))$. The computation complexity of finding set intersections in line 10 takes $O(|E|!)$. Line 13-17 produces trivial complexity compared to the others. The while loop in line 4-17 runs $|AUTH| < |V|^2$ times. Hence, the overall complexity of Algorithm 4.1 is $O(|V|^4 \times (|E|!))$.

The asymptotic complexity of the current approach is high, especially because computation of all possible simple paths between any pair of vertices in RG gives the ultimate lower bound. However, RG can be a sparse one. Also, it can be easily noticed that pre-computing and storing all possible simple paths between any pairs in RG regardless of the AUTH can effectively reduce the computation time inside the loop. Moreover, for many such practical problems heuristic solutions are often effective. Later in this study, it has been discussed that our ReBAC rule structure is not the most general one. Feasibility algorithm can certainly change based on the variety of ReBAC rule structures. Therefore, overall complexity of determining ReBAC policy mining feasibility can vary based on such factors. A detailed study of these is out of scope of this chapter.

The correctness proof of Algorithm 4.1 is as follows.

Theorem 5. *Given a RREP-0 instance as in Def. 30, a suitable RuleSet exists iff Algorithm 4.1 generates the* $Rule_{op}$.

Proof:

Assume, Algorithm 4.1 generates the $Rule_{op}$. According to Algorithm 4.1, for each $(a, b) \in AUTH$, all possible paths from a to b in RG are searched over to find the collection of pathLabel(p) where p is a simple path from a to b, such that there exists no unauthorized tuple $(c, d) \in V \times V \setminus AUTH, c \neq d$ where the collection of pathLabel(q) is a superset of the collection of pathLabel(p), where q is a simple path from c to d in RG. In Algorithm 4.1, $Rule_{op}$ consists of disjunctions of such conjunction of the collection of pathLabel(p), generated for each $(a, b) \in AUTH$. By the definition of checkAccess in Def. 29, the generated $Rule_{op}$ evaluates to true for each $(a, b) \in AUTH$ while denying all $(c, d) \in V \times V \setminus AUTH, c \neq d$. Hence, $Rule_{op}$ constitutes a suitable

RuleSet.

To prove the opposite direction, assume a suitable RuleSet $Rule'_{op}$ constituted by Def. 28 exists. Therefore, by the definition of RREP-0, $Rule'_{op}$ evaluates to true for each $(a, b) \in AUTH$ while denying all unauthorized tuple $(c, d) \in V \times V \setminus AUTH, c \neq d$. By the procedure of $Rule'_{op}$ evaluation provided in Def. 29, there exists at least a conjunctive term in $Rule'_{op}$ which is true for a $(a, b) \in AUTH$ where for all pathLabelExprs in the corresponding conjunctive term, there exists a simple path p from a to b in RG such that pathLabel(p) = pathLabelExpr. According to Algorithm 4.1, for each $(a, b) \in AUTH$, all possible paths from a to b in RG are searched over to find such conjunction of the collection of pathLabel(p) and $Rule_{op}$ consists of disjunction of such conjunctions, generated for each $(a, b) \in AUTH$. Thereby, Algorithm 4.1 generates the feasible status and $Rule_{op}$, where each conjunctive term denoted by t' in $Rule'_{op}$ must have at least a conjunctive term t in $Rule_{op}$ where the pathLabels in t' are a subset of the pathLabels in t. Hence, the claim holds in both directions and Theorem 5 is proved.

RREP-0 is the core of our ReBAC feasibility analysis. An example of ReBAC rule generation is discussed in Section 4.6.

4.4 Variations of ReBAC Ruleset Existence Problem

By definition of RREP-0 in Def. 30, there are three key factors which affect the feasibility detection process: i) the authorization relation AUTH, ii) $Rule_{op}$ structure, and iii) RG. For example, an AUTH relation can be symmetric or asymmetric, RG can be directed or undirected, and the $Rule_{op}$ specification grammar can be modified to add more or less expressive power. In this section, we consider some RREP variations focusing on ReBAC rule structure.

4.4.1 Proposed RREP Variations

The following discussion proposes four variations of RREP. According to Def. 25, the given RG is a directed labeled graph. Therefore, Algorithm 4.1 can work with directed RG only. Given an

undirected relationship graph $RG^{\gamma} = (V, E^{\gamma}, \Sigma)$, an equivalent directed labeled relationship graph RG = (V, E, Σ) can be generated by enhancing the set of edges. For each edge $(a, b, \sigma) \in E^{\gamma}$, symmetric edges (a, b, σ) and (b, a, σ) are added to E. For each $(u, v) \in AUTH$, symmetric authorization tuples (u, v) and (v, u) are added to updated AUTH relation as well. It is evident that the undirected RG along with undirected AUTH can be reduced to core RREP-0 and Algorithm 4.1 can be deployed to solve the feasibility detection. Thus it suffices to consider directed RG.

Before proceeding to the other variations of RREP, three extended sets of relationships are defined as follows for a given Σ.

- $\overline{\Sigma} = \{\overline{\sigma} | \sigma \in \Sigma\}$. For each relation type specifier $\sigma \in \Sigma$, $\overline{\sigma}$ denotes "no σ relation". Therefore, $\overline{\Sigma}$ is the set of non-relationship type specifiers in RG.
- $\Sigma^{-1} = \{\sigma^{-1} | \sigma \in \Sigma\}$. For each relation type specifier $\sigma \in \Sigma$, σ^{-1} denotes "inverse σ relation". Therefore, Σ^{-1} is the set of inverse relation type specifiers in RG.
- $\overline{\Sigma}^{-1} = \{\overline{\sigma}^{-1} | \overline{\sigma} \in \overline{\Sigma}\}$. Here, $\overline{\Sigma}^{-1}$ denotes the set of non-relationship inverse relation type specifiers in RG.

The inverse non-relationship specifier $\overline{\sigma^{-1}}$ is not considered, since it is equivalent to $\overline{\sigma}^{-1}$ and hence redundant.

Theorem 6. *The inverse non-relationship specifier $\overline{\sigma^{-1}}$ is not considered, since it is equivalent to $\overline{\sigma}^{-1}$ and hence redundant.*

Proof

Given a $\sigma \in \Sigma$, $\overline{\sigma^{-1}}$ is called the inverse non-relationship of σ. We show that $\overline{\sigma}^{-1} \equiv \overline{\sigma^{-1}}$. Fig. 4.2 shows a sequence of equivalences going from top to bottom, or vice versa, which establish this. Relationships that cannot exist are shown in dotted lines while relationships that must exist are shown in solid lines. From top to bottom, a relationship $\overline{\sigma^{-1}}$ from a to b precludes a relationship of σ^{-1} from a to b, which in turn precludes a relationship of σ from b to a. Thereby there is a non-relationship $\overline{\sigma}$ from b to a, and finally its inverse $\overline{\sigma}^{-1}$ from a to b. The argument in reverse holds from bottom to top.

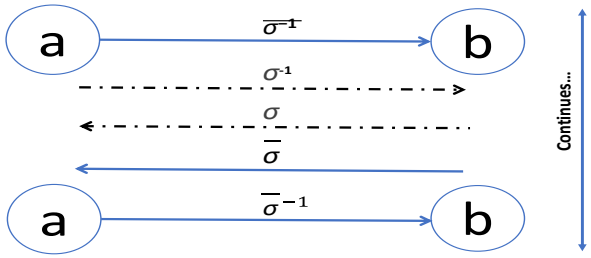

Figure 4.2: Given a $\sigma \in \Sigma$, $\overline{\sigma}^{-1} \equiv \overline{\sigma^{-1}}$

Table 4.1: Path variations in RG

Characteristics	SCP	SPP	SCPP
$(a, b, \sigma) \rightarrow (a, b, \sigma) \in E, \sigma \in \Sigma$	✓	✓	✓
$(a, b, \overline{\sigma}) \rightarrow (a, b, \sigma) \notin E, \overline{\sigma} \in \overline{\Sigma}$	✓		✓
$(a, b, \sigma^{-1}) \rightarrow (b, a, \sigma) \in E, \sigma^{-1} \in \Sigma^{-1}$		✓	✓
$(a, b, \overline{\sigma}^{-1}) \rightarrow (b, a, \sigma) \notin E, \overline{\sigma}^{-1} \in \overline{\Sigma}^{-1}$			✓

There is no redundancy amongst $\overline{\sigma}, \sigma^{-1}$ and $\overline{\sigma}^{-1}$, as we will see in Section 4.6.

RREP-0 uses simple path definition in RG. In order to specify extensions to RREP-0, three path variations in RG are defined as follows utilizing the extended relation types defined above.

Definition 31. Path Variations in RG

The definition of (simple) linked sequence of vertices in Def. 26 is extended to include the extended symbols in $\overline{\Sigma}$, Σ^{-1} and $\overline{\Sigma}^{-1}$, in addition to Σ. The definition of (simple) path in Def. 27 is extended as summarized in Table 4.1 to give the following three extended notions of path.

i) Simple Complementary Path (SCP) allows symbols from Σ and $\overline{\Sigma}$ respectively requiring the triple to be an edge or not an edge as indicated in the top two rows of Table 4.1.

ii) Simple Permissive Path (SPP) allows symbols from Σ and Σ^{-1} respectively requiring the triple to be an edge or the inverse of an edge as in the first and third rows of Table 4.1.

Table 4.2: RREP variations

(a) RREP-0	(b) RREP-1	(c) RREP-2	(d) RREP-3
RuleSet as in Def. 28	$edgeLabel ::= \sigma \mid \overline{\sigma}$	$edgeLabel ::= \sigma \mid \sigma^{-1}$	$edgeLabel ::= \sigma \mid \overline{\sigma} \mid \sigma^{-1} \mid \overline{\sigma}^{-1}$
checkAccess as in Def. 29	simple path is replaced by SCP	simple path is replaced by SPP	simple path is replaced by SCPP

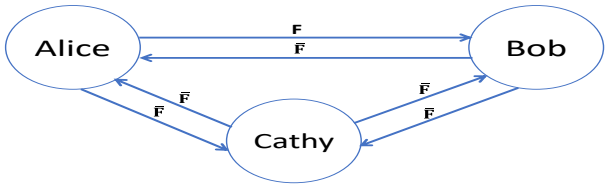

Figure 4.3: RG of Fig. 4.1 enhanced with non-relationship edges.

iii) Simple Complementary Permissive Path (SCPP) allows symbols from Σ, $\overline{\Sigma}$, Σ^{-1} and $\overline{\Sigma}^{-1}$ respectively requiring the triple to be an edge, not an edge, the inverse of an edge or the inverse of a "not an edge" as in the four rows of Table 4.1.

Based on the three path definitions introduced above, three variations of RREP problem, named as RREP-1, RREP-2, and RREP-3 are defined as follows.

Definition 32. RREP-1, RREP-2, and RREP-3

Given the definition of RREP-0 as in Def. 30, the definitions of RREP-1, RREP-2, and RREP-3 are similar, except for distinctions noted in Table 4.2.

Table 4.2 describes the distinctions of RREP-1 to 3 in terms of comparison with RREP-0 features. Table 4.2 shows that RREP-1 to 3 vary from RREP-0 based on two related aspects: RuleSet and checkAccess definitions. Row 1 of Table 4.2 shows that, the $Rule_{op}$ grammar specified for RREP-1 to 3 vary in edgeLabel definitions only, compared to RREP-0 RuleSet definition as in Def. 28. Row 2 of Table 4.2 shows that $Rule_{op}$ of RREP-1 to 3 uses the same evaluation criteria compared to RREP-0, except simple path is changed to SCP, SPP and SCPP respectively.

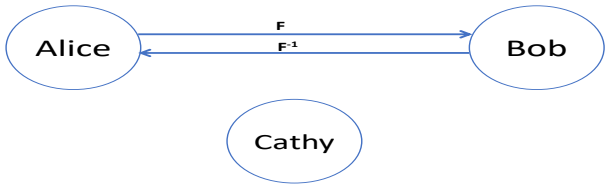

Figure 4.4: RG of Fig. 4.1 enhanced with inverse edges.

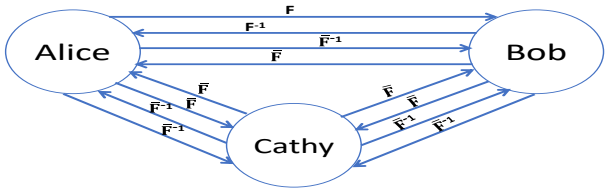

Figure 4.5: RG of Fig. 4.1 enhanced with non-relationship, inverse and non-relationship inverse edges.

4.4.2 Reduction of RREP Variations

The major difference between RREP-0 and the proposed variations RREP-1 to RREP-3 is that simple path definition in RREP-0 consists of given edges in RG, whereas SCP, SPP and SCPP bring additional "virtual edges" into consideration. We can reduce the enhanced path definitions of SCP, SPP and SCPP to the traditional path definition by enhancing the original RG with these virtual edges. Given the RG of Fig. 4.1, its enhancements with additional edges for SCP, SPP and SCPP are respectively shown in Figs. 4.3, 4.4 and 4.5. These enhancements are formally stated as follows.

Definition 33. Enhancements of RG

Given a directed labeled relationship graph RG = (V, E, Σ), let

- $\overline{E} = \{(u, v, \overline{\sigma}) \mid u \neq v \wedge (u, v, \sigma) \notin E\}$

  These are called non-relationship edges.

- $E^{-1} = \{(u, v, \sigma^{-1}) \mid u \neq v \wedge (v, u, \sigma) \in E\}$

These are called inverse edges.

- $\overline{E}^{-1} = \{(u, v, \overline{\sigma}^{-1}) | u \neq v \wedge (v, u, \sigma) \notin E\}$

  These are called non-relationship inverse edges.

The enhanced RG, denoted RG_{E}, is defined as follows:

- For RREP-1: $RG_{\mathrm{E}} = (V, E \cup \overline{E}, \Sigma \cup \overline{\Sigma})$

Note that $\mathrm{RG_E}$ imposes some consistency requirements such as (u, v, σ) is an edge in $\mathrm{RG_E}$ iff $(u, v, \overline{\sigma})$ is not an edge in $\mathrm{RG_E}$.

The lemma below follows trivially from the definitions.

Lemma 10. *There is an SCP (respectively SPP, SCPP) p from u to v with pathLabel(p) in RG iff there is a simple path p from u to v in* RG_{E} *for RREP-1 (respectively RREP-2, RREP-3) with pathLabel(p).*

It follows that Algorithm 4.1 for RREP-0 with correspondingly enhanced RG can be used to solve the feasibility detection problem for RREP-1, RREP-2 and RREP-3 as well.

4.4.3 Limitation of RREP-0 to RREP-3

It is easy to construct examples that are beyond the scope of the variations discussed above. In the RGs of both Fig. 4.6 and Fig. 4.7, there are two simple paths from Alice to Ray with path labels "F.F" and "F.F.F". However, there is a significant difference between the two RGs. In Fig. 4.6 the simple paths from Alice to Ray are disjoint with respect to their edges, while this is not so for Fig. 4.7. Specification of disjoint paths is not possible in our rule structure variations. In the most general case any computable property of RG can be utilized in the rule structure.

A ReBAC policy language for user to user relationship is presented in [17]. Although [17] offers different rule structures for accessing user, target user as well as system administrator views, a basic comparative study between the rule set structure of our work and the ReBAC policy presented in [17] is as follows.

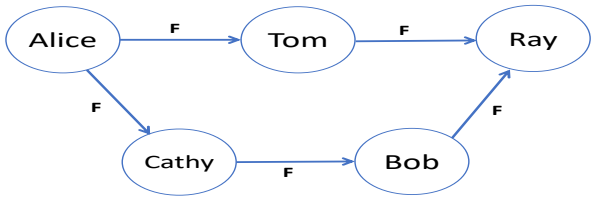

Figure 4.6

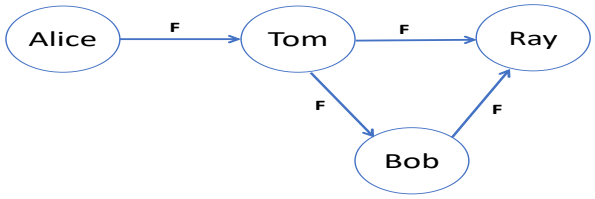

Figure 4.7

1. In [17], each pathLabelExpr is limited by the maximum number of edges allowed in the path, specified as hopcount. Our rule structure does not allow such numeric value on edge count in RG. Moreover, [17] offers negative pathLabelExpr, that means an entire relationship pattern that must not exist from accessing user to target user in the RG. In our work, allowing non-relationship edges accomplish the fact of traversing the graph in "not in a relationship" directions, however, its semantics is completely different.

2. Repeating a relationship pattern unlimited (*) or 0/1 times (?) has been included in [17]. Our ReBAC policy can accomplish the similar task by repeating the rule expressions as many times as desired. Note that infinite repetitions are not possible for simple paths in a finite graph.

3. The rule evaluation in [17] can start from a particular user as noted in the rule, but our ReBAC policy evaluation starts from any node in RG, therefore, can be referred as system policy. For both of the works, pathLabelExprs are constituted by using disjunction and conjunction operators.

From the discussions above, it can be summarized that, our rule structure lacks some features as compared to [17] such as hopcount on the pathLabelExpr, enhances a few such as allowing complementary and permissive path in RG, and has similar structure such as use of disjunction and conjunction of pathLabelExpr.

4.5 Proposed Infeasibility Solutions

In this section we propose a solution to infeasibility in RREP-0 and illustrated by examples. Other possible direction of solution approaches and limitations will be discussed briefly.

4.5.1 Proposed Infeasibility Correction

Given a RREP-0 instance as in Def. 30, if no suitable RuleSet exists (i.e., Algorithm 4.1 returns infeasible result) we say there is an infeasibility problem. In such cases we can make a suitable RuleSet generation possible by adding new relationships as follows.

i) Select a symbol $op \notin \Sigma$.

ii) Add the path expression op as a disjunction to the generated $Rule_{op}$ by Algorithm 4.1 to construct $Rule_{op} \vee op$.

iii) For each $(u, v) \in failedAuthList$ add an edge (u,v,op) to E in RG.

Theorem 7. *The infeasibility correction solution above is correct for the modified RREP-0 problem with modified RG and* Σ.

Proof:

For each $(u, v) \in AUTH$, where Algorithm 4.1 fails to generate the rule, the proposed solution above adds an edge from u to v in RG with edge label op. It is trivial that a simple path of length 1 with pathLabelExpr op thereby exists in the modified RG for each such $(u, v) \in failedAuthList$, generated by Algorithm 4.1. Therefore, the pathLabelExpr op turns true for each such infeasible authorization tuples only.

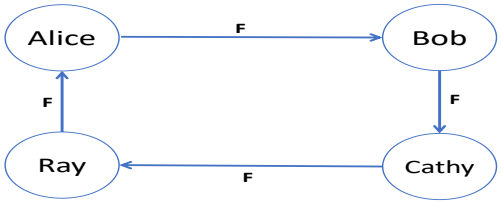

Figure 4.8: RREP-0 infeasibility example.

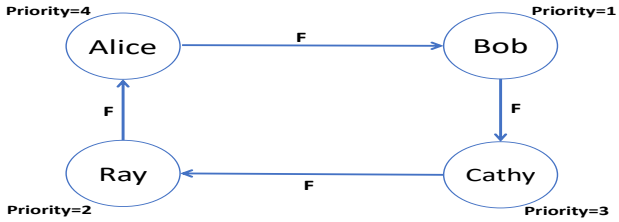

Figure 4.9: Adding "priority" attribute to Fig. 4.8.

Fig. 4.8 presents an RG the set of users $V = \{Alice, Bob, Cathy, Ray\}$, the set of edges E = $\{(Alice, Bob, F), (Bob, Cathy, F), (Cathy, Ray, F), (Ray, Alice, F)\}$, and the set of relation type specifiers, $\Sigma = \{F\}$. Let AUTH = $\{(Alice, Bob), (Cathy, Ray)\}$. According to the RuleSet structure given in Def. 28, RREP-0 fails since there exists a single simple path from Alice to Bob, where path label is F. However, "F" is also true for (Alice, Bob), (Bob, Cathy), (Cathy, Ray), and (Ray, Alice). The same scenario occurs while finding rule for (Cathy, Ray). Therefore, the given AUTH is concluded as infeasible by Algorithm 4.1, and failedAuthList contains both (Alice, Bob) and (Cathy,Ray). According to the solution above, two additional edges (Alice, Bob,op) and (Cathy,Ray,op) are added to E, and Σ is updated to $\{F, op\}$. The generated $Rule_{op}$ is op.

4.5.2 Alternate Infeasibility Correction

An alternate approach to infeasibility correction is to add an attribute named "priority" to each vertex in the RG, illustrated by example as follows. Consider the solution provided in Fig. 4.9, given the prior infeasibility example: AUTH = $\{(Alice, Bob), (Cathy, Ray)\}$ and RG is as shown

in Fig. 4.8. Each user vertex in the given RG in Fig. 4.8 has been assigned a positive integer priority value, and the ordered sequence of vertex priority values associated with the path p from vertex a to b in RG is the same order followed by the vertices through the path p. For example, ordered sequence of vertex priority values associated with the path from Alice to Ray in Fig. 4.9 is $\langle 4, 1, 3, 2 \rangle$. The $Rule_{op}$ given in Def. 28 is modified in order to accommodate the use of priority value as follows:

$Rule_{op} ::= Rule_{op} \vee Rule_{op} | pathRuleExpr$

$pathRuleExpr ::= pathRuleExpr \wedge pathRuleExpr \ | (pathLabelExpr, priorityOrder)$

$priorityOrder ::= > \ | < | \phi$

$pathLabelExpr ::= pathLabelExpr.pathLabelExpr | edgeLabel$

$edgeLabel ::= \sigma, \sigma \in \Sigma$

where >,<, and ϕ represent increasing, decreasing and don't care orders, respectively, and pathRuleExpr consists of conjunction of (pathLabelExpr, priorityOrder) pairs.

The evaluation procedure of $checkAccess_{ReBAC}$(a:V, b:V) in a ReBAC system with the specified $Rule_{op}$ is as follows:
(i) for each (pathLabelExpr, Order) pair in $Rule_{op}$ substitute True if there exists a simple path p from a to b in RG with path label pathLabelExpr where the ordered sequence of vertex priority values associated with path p follows the priorityOrder order, otherwise substitute False, (ii) evaluate the resulting boolean expression.

Let's recall the AUTH = $\{(Alice, Bob), (Cathy, Ray)\}$ noted earlier for Fig. 4.8. According to the proposed $Rule_{op}$ structure, the generated $Rule_{op} = (F, <)$ solves the infeasibility because the simple path labeled F from Alice to Bob follows the decreasing order as $4 > 2$. The same case occurs for (Cathy, Ray) since $3 > 2$, whereas (Bob, Cathy) and (Ray, Alice) do not.

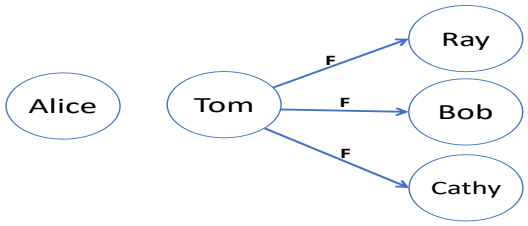

Figure 4.10

4.5.3 Limitations of Current Infeasibility Solution

The infeasibility solution provided in Section 4.5 adds only a single pathLabelExpr "op" to the $Rule_{op}$, regardless of the number of infeasible tuples in the AUTH, adding $|AUTH|$ number of additional edges in RG in the worst case. For example, given the RG in Fig. 4.10 and the set of authorization tuples $\{(Alice, Ray), (Alice, Bob), (Alice, Cathy)\}$, this solution adds three edges with label op originating from Alice to Bob, Ray, and Cathy, respectively. Therefore, the $Rule_{op}$ is "op". However, a solution fewer added edges can be obtained for the given AUTH by adding a single edge from Alice to Tom labeled as "op". It is clearly evident that the edge from Alice to Tom creates simple paths from Alice to Ray, Bob, and Cathy. Therefore, the possible $Rule_{op}$ for the given AUTH is "op.F". This demonstrates the trade-off between minimum size of rule and adding minimum number edges in RG to correct infeasibility. The solution of Subsection 4.5.1 keeps the given RG unchanged while adding new relationship edges to RG. An alternative approach could be to remove some edges from the given RG.

4.6 Use Cases and Implementation

In this section we present case studies to show the relative power of the rule structures of RREP variations defined in this study. We also discuss the need for rule optimization.

Consider the RG shown in Fig. 4.11(a), along with its inverse, non-relationship and non-relationship inverse edges shown in Figs. 4.11(b), 4.12(a) and 4.12(b). For different values of AUTH we get different feasibility results as follows, where we understand that Algorithm 4.1 will

be run with correspondingly enhanced RGs (i.e., Fig. 4.11(a) for RREP-0, union of Figs. 4.11(a) and 4.12(a) for RREP-1, union of Figs. 4.11(a) and 4.11(b) for RREP-2, and union of Figs. 4.11(a), 4.11(b), 4.12(a) and 4.12(b) for RREP-3).

1. Let AUTH = {(Ray,Cathy), (Bob,Cathy)}. Then Algorithm 4.1 will return success for RREP-0, RREP-1, RREP-2 and RREP-3. Note that feasibility of RREP-0 always implies feasibility of RREP-1, RREP-2 and RREP-3 since the simple path of RREP-0 is included in the enhanced path definitions of the latter. The rule returned for RREP-0 and RREP-2 is F$\vee$F which is logically equivalent to F. The rules generated for RREP-1 and RREP-3 are more complex due to the increased number of paths in the enhanced RGs.

2. Let AUTH = {(Cathy,Ray), (Cathy,Bob)}. For RREP-0 and RREP-1 Algorithm 4.1 will return failure. For RREP-2 it will return $F^{-1} \vee F^{-1}$. The formula for RREP-3 is more complex.

3. Let, AUTH = {(Alice, Bob), (Alice, Cathy), (Alice, Ray), (Bob, Alice), (Bob, Ray), (Cathy, Alice), (Cathy, Bob), (Cathy,Ray), (Ray, Alice), (Ray,Bob)}. For RREP-0 and RREP-2 Algorithm 4.1 will return failure. For RREP-1 and RREP-3 it will return success with complex formulae due to the multiplicity of paths in the enhanced RGs.

4. Let's consider, AUTH = {(Ray,Cathy), (Bob,Cathy), (Cathy,Ray), (Cathy,Bob), (Alice, Cathy)}. For RREP-0, RREP-1 and RREP-2 Algorithm 4.1 will return failure. For RREP-3 it will return success with a complex formula which would logically reduce to $\overline{F}^{-1}.\overline{F}^{-1}.F \vee F^{-1}$.

These examples establish that the rule structure of RREP-3 is strictly more expressive than RREP-0, RREP-1 and RREP-2. Note that RREP-0 is the weakest as argued above. RREP-1 and RREP-2 are incomparable.

The generated rule may contain unnecessary path labels in conjunctive terms if all possible path labels are being used. Therefore, a few simple rule optimization techniques are used in the implementation. As stated in Algorithm 4.1, for any tuple (a,b) in AUTH, all possible path labels from a to b are AND'ed to form the conjunctive term after determining the feasibility. Instead of

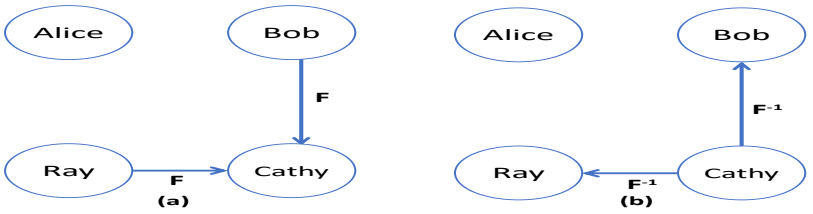

Figure 4.11: (a) Given RG, (b) Inverse edges for RG of Fig. 4.11(a)

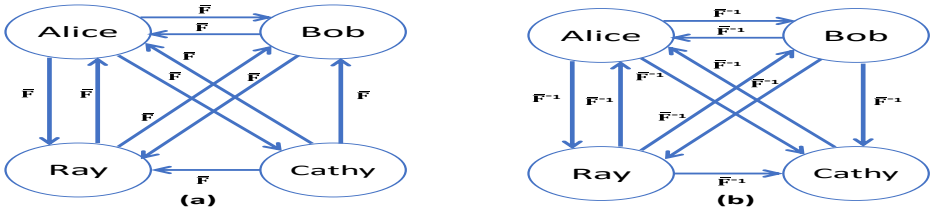

Figure 4.12: (a) Non-relationship edges for RG of Fig. 4.11(a), (b) Non-relationship inverse edges for RG of Fig. 4.11(a)

using all possible path labels, the smallest possible subset of those is used to form the conjunctive term such that it does not evaluate to true for any unauthorized tuple. For example, given the RG in Fig.4.11(a), and an EAS where V is identical and set of authorization relations AUTH = $\{(Alice, Ray), (Alice, Bob)\}$, the $Rule_{op}$ computed using RREP-3 by Algorithm 4.1 comprises a conjunction of 24 terms as follows:

$\overline{F}.F^{-1} \wedge \overline{F}^{-1} \wedge \overline{F}.\overline{F} \wedge \overline{F}^{-1}.\overline{F} \wedge \overline{F}.\overline{F}^{-1} \wedge \overline{F}^{-1}.\overline{F}^{-1}.F^{-1} \wedge \overline{F}^{-1}.\overline{F}.\overline{F} \wedge \overline{F}^{-1}.F^{-1} \wedge \overline{F}^{-1}.F^{-1}.\overline{F} \wedge$
$\overline{F}.F^{-1}.\overline{F}^{-1} \wedge \overline{F}.\overline{F}^{-1}.\overline{F} \wedge \overline{F}^{-1}.F^{-1}.\overline{F}^{-1} \wedge \overline{F}.\overline{F}.\overline{F} \wedge \overline{F}.\overline{F}.\overline{F}^{-1} \wedge \overline{F}^{-1}.\overline{F}.\overline{F}^{-1} \wedge \overline{F}^{-1}.\overline{F}^{-1}.\overline{F} \wedge$
$\overline{F}.F^{-1}.\overline{F} \wedge \overline{F}.F.\overline{F} \wedge \overline{F} \wedge \overline{F}^{-1}.F.F^{-1} \wedge \overline{F}.\overline{F}^{-1}.F^{-1} \wedge \overline{F}^{-1}.F.\overline{F} \wedge \overline{F}^{-1}.\overline{F}^{-1} \wedge \overline{F}.F.F^{-1}$

Both tuples (Alice, Bob) and (Alice, Ray) in AUTH would generate this conjunction since they have the same set of path labels. After applying the specified smallest possible subset of path labels in a conjunctive term technique, the specified $Rule_{op}$ turns into significantly smaller rule, given by $\overline{F}.F^{-1}$. Another way of rule minimization is: after completion of rule generation, a conjunctive term in the generated rule, say c1, removes all conjunctive terms c2 in the rule if all path labels in c1 are included in c2. The Java implementation of feasibility detection along with

the described rule minimization techniques can be provided upon request.

CHAPTER 5: ON FEASIBILITY OF ATTRIBUTE-AWARE RELATIONSHIP-BASED ACCESS CONTROL POLICY MINING

Relationship-Based Access Control (ReBAC) emerged from the access control requirements of Online Social Networks, and expresses authorization policy in terms of various relationship parameters such as type and depth, whereas Attribute-Based Access Control (ABAC) has been motivated by its generalized structure and versatility in access control policy specification through attributes of user, resource, environment, etc. Although combination of ABAC and ReBAC, such as Attribute-aware ReBAC (AReBAC), have previously been defined and shows additional expressive power compared to standalone ABAC and ReBAC, feasibility analysis of AReBAC policy mining is still unexplored. This chapter studies whether conversion to AReBAC system is possible from an Enumerated Authorization System (EAS) given supporting attribute and relationship data. Attribute-aware ReBAC Ruleset Existence Problem (ARREP) has been introduced formally for the first time, and solved algorithmically along with complexity analysis. In case of infeasibility, notions of equivalent and approximate solutions are developed. Directions for future enhancements are also discussed.

Significant portion of this chapter has been published at the following venue [12].

- Shuvra Chakraborty and Ravi Sandhu, On Feasibility of Attribute-Aware Relationship-Based Access Control Policy Mining. In Proc. 33rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec), Virtual Event, July 19-20, 2021.

5.1 Motivation

Although both ReBAC and ABAC are powerful, flexible and comparable [2] in expressing authorization policies, relying solely on one is often insufficient. An example case will be used in order to compare ABAC policy presented in [13] and ReBAC policy in [11] with the proposed AReBAC policy in this study.

Consider the relationship graph with attributes in Fig. 5.1 where the set of users V = {Alice,

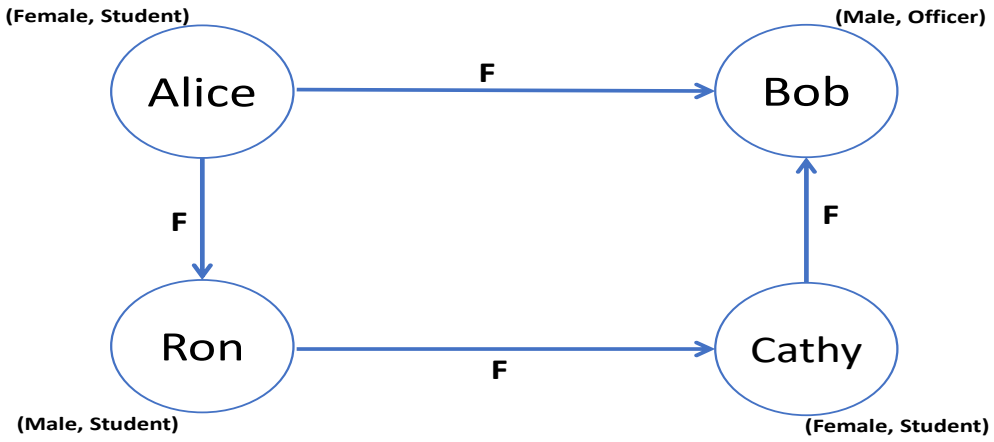

Figure 5.1: Example ARG with (Gender, Profession) user attribute values

Bob, Cathy, Ron}, the set of user attribute function names, VA = $\{Gender, Profession\}$, the set of relationship specifiers, Σ = {F} where "Friendship" is abbreviated as "F", and the set of edges E = { (Alice, Ron, F), (Alice, Bob, F), (Ron, Cathy, F), (Cathy, Bob, F) }.

1. ReBAC policy can express the authorization state (Alice, Bob) whereas only ABAC policy cannot. ABAC rule fails because Alice and Cathy have the same attribute value combination. The generated ReBAC rule is "F.F.F".

2. ABAC policy can express the authorization state (Ron, Bob) whereas only ReBAC policy cannot. ReBAC rule fails because there is only one path labeled "F.F" from Ron to Bob which is satisfied by unauthorized pair, such as, (Alice, Cathy). The generated ABAC rule is, "Gender(u)=Male $\wedge$ Profession(u)=Student $\wedge$ Gender(v)=Male $\wedge$ Profession(v)=Officer".

3. Authorization state (Alice, Ron), cannot be expressed by both ABAC and ReBAC. ABAC rule fails because Alice and Cathy have the same attribute value combination. ReBAC rule fails because the only path label "F" is satisfied by other unauthorized pairs, such as (Alice, Bob).

These example cases shows that, there are certain conditions where ABAC and ReBAC can

fail, either singly or both. Later on, the same example will be used to show that AReBAC is more expressive than ABAC [13] and ReBAC [11]. Additionally, it can be clearly observed that, if entity ids are allowed, AReBAC policy will never fail (such as [6]). However, imposing this condition conflicts with core principles of ABAC and ReBAC. Therefore, the AReBAC policy specification in this study checks whether the target access control system could be generated avoiding explicit use of unique entity id.

5.2 Preliminaries

In this section, some preliminaries of Attribute-aware ReBAC RuleSet Existence Problem will be noted and rest of the chapter will repeatedly use these definitions.

A user/subject is an entity who performs operation on a resource/object. The set of users is represented by U. A user requests to perform an operation on another user. An operation is an action performed by a user on another user. The set of operations in the system is represented by OP. Without loss of generality it is assumed that OP is a singleton given by $\{op\}$, since each operation has its specific policy or rules. An access request is a tuple $\langle u, v\rangle$ where user u is asking permission to perform operation op on user v where $u, v \in U, op \in OP, u \neq v$. An access request is either granted or denied, based on the access control policy. In any access control system, a logical construct is required to decide the outcome of an access request. The logical construct is formally defined as, $checkAccess\colon U \times U \rightarrow \{True, False\}$, where the result $True$ grants access while $False$ denies it.

5.2.1 Source Access Control System

As mentioned before, this chapter studies Attribute-aware ReBAC RuleSet Existence problem with a single type of source access control system.

We define a simple authorization system, EAS as follows:

Definition 34. Enumerated Authorization System (EAS)

An EAS is a tuple $\langle$U, AUTH, $checkAccess_{EAS}\rangle$ where, U is the finite sets of users and $AUTH \subseteq$

$U \times U$, is a specified authorization relation where

$$checkAccess_{EAS}(u, v) \equiv (u, v) \in AUTH$$

For example, given U = $\{Alice, Bob\}$ and OP=$\{readData\}$, Bob can read Alice's data iff (Bob, Alice) belongs to AUTH.

5.2.2 Target Access Control System

In this chapter, the target access control system is Attribute-aware ReBAC system. Before defining the Attribute-aware ReBAC RuleSet Existence Problem, a complete specification of Attribute-aware ReBAC policy as well as the rule evaluation procedure is necessary

In order to define an Attibute-aware ReBAC system, the key component is Attribute-aware Relationship Graph (ARG), which is defined as follows.

Definition 35. Attribute-aware Relationship Graph (ARG)

The Attribute-aware Relationship Graph ARG = (V, VA, VA-RangeSet, UATTValue, EA, EA-RangeSet, E) is a directed labeled graph where,

a. V is the set of vertices in ARG, representing the set of users in the system.

b. VA is the finite set of atomic user attribute function names $\{va_1, va_2, ..., va_m\}$.

c. For each $va_i \in VA$, Range(va_i) specifies a finite set of atomic values for user attribute va_i. VA-RangeSet = $\{(va_i, value) | va_i \in VA \wedge value \in Range(va_i)\}$.

d. UATTValue denotes the user attribute value assignments. UATTValue = $\{UATTValue_{va_i} | va_i \in VA\}$ where $UATTValue_{va_i} : V \rightarrow Range(va_i)$. For convenience, we understand va_i(a) to denote $UATTValue_{va_i}(a)$, that is the attribute value assignment of an actual user a for attribute va_i.

e. EA is the finite set of edge attribute function names, $\{ea_1, ea_2, ..., ea_n\}$.

f. For each $ea_i \in EA$, Range(ea_i) specifies a finite set of atomic values for edge attribute ea$_i$. EA-RangeSet = $\{(ea_i, value) | ea_i \in EA \wedge value \in Range(ea_i)\}$.

g. $E \subseteq V \times V \times Range(ea_1) \times Range(ea_2) \times ... \times Range(ea_n)$ is a finite set of directed edges where, an edge $(u, v, \sigma_1, \sigma_2, ..., \sigma_n) \in E$, $u \neq v$, represents the relations $\sigma_1, \sigma_2, ..., \sigma_n$ from user $u \in V$ to $v \in V$ in ARG where $\sigma_1 \in Range(ea_1), \sigma_2 \in Range(ea_2), ..., \sigma_n \in Range(ea_n)$.
Note: For a directed edge e from vertex a to vertex b in ARG, ea$_i$(e) specifies the associated edge attribute value assignment for $ea_i \in EA$.

Fig. 5.1 presents an ARG where the set of users V = {Alice, Bob, Cathy, Ron}, the set of user attribute function names, VA = $\{Gender, Profession\}$, the set of edge attribute function names, EA = $\{Relation - type\}$, and the set of edges E = { (Alice, Ron, F), (Alice, Bob, F), (Ron, Cathy, F), (Cathy, Bob, F) }. The user and edge attribute value assignments are shown in Fig. 5.1. The notion of a path in an ARG is defined as follows:

Definition 36. Path in ARG

Given ARG as in Def. 35 and a vertex pair $(u, v) \in V \times V$ where $u \neq v$, a path from u to v is a sequence of edges where the terminating (i.e., second) vertex of each edge is same as the starting (i.e., first) vertex of the next edge given by $\langle (u, v_i, \sigma_{w1}, \sigma_{w2}, ..., \sigma_{wn}), (v_i, v_j, \sigma_{x1}, \sigma_{x2}, ..., \sigma_{xn}), ..., (v_k, v_l, \sigma_{y1}, \sigma_{y2}, ..., \sigma_{yn}), (v_l, v, \sigma_{z1}, \sigma_{z2}, ..., \sigma_{zn}) \rangle$, where

a. $u, v_i, v_j, ..., v_k, v_l, v \in V$

b. $\sigma_{w1}, \sigma_{x1}, ..., \sigma_{y1}, \sigma_{z1} \in Range(ea_1), \sigma_{w2}, \sigma_{x2}, ..., \sigma_{y2}, \sigma_{z2} \in Range(ea_2)$, ..., $\sigma_{wn}, \sigma_{xn}, ..., \sigma_{yn}, \sigma_{zn} \in Range(ea_n)$.

A path p from u to v is said to be simple iff $u, v_i, v_j, ..., v_k, v_l, v \in V$ are distinct. The length of p, denoted by $|p|$, is the number of edges in the path. The attribute aware path label of the path p from u to v, denoted by $pathLabel_{att}(p)$, is
$(va_1(u), va_2(u), ..., va_m(u)).(\sigma_{w1}, \sigma_{w2}, ..., \sigma_{wn}).(va_1(v_i), va_2(v_i), ..., va_m(v_i)).(\sigma_{x1},$

$\sigma_{x2}, ..., \sigma_{xn}).(va_1(v_j), va_2(v_j), ..., va_m(v_j)).....(va_1(v_k), va_2(v_k), ..., va_m(v_k)).(\sigma_{y1},$
$\sigma_{y2}, ..., \sigma_{yn}).(va_1(v_l), va_2(v_l), ..., va_m(v_l)).(\sigma_{z1}, \sigma_{z2}, ..., \sigma_{zn}).(va_1(v), va_2(v), ...,$
$va_m(v))$.

Clearly, $pathLabel_{att}(p)$ is a string, consisting of concatenated tuples of vertex and edge attribute value assignments, traversed in the same order as the vertices and edges appear in path p. Note that, the vertex and edge attribute values follow specific orders, given by $\langle va_1, va_2, ..., va_m \rangle$ and $\langle ea_1, ea_2, ..., ea_n \rangle$, respectively. For sth edge in path p where $1 \leq s \leq |p|$, starting vertex, edge, and terminating vertex attribute value assignments are represented by $(2 \times s - 1)$th, $(2 \times s)$th, and $(2 \times s + 1)$th tuples in $pathLabel_{att}(p)$, respectively.

Given ARG in Fig. 5.1, the only path p from Cathy to Bob is $\langle (Cathy, Bob, F) \rangle$ with $pathLabel_{att}(p)$ = $(Female, Student).(F).(Male, \textit{Officer})$. Henceforth, we understand path to mean simple path.

Definition 37. Attribute aware ReBAC policy

An Attribute aware ReBAC policy, POL_{AAR} is a tuple, given by $\langle$ OP, VA, EA, RuleSet$\rangle$ where,

a. OP, VA, and EA are as defined in Def. 35.

b. RuleSet is a set of rules where, for each operation $op \in OP$, RuleSet contains a rule $Rule_{op}$. Each $Rule_{op}$ is specified using the grammar below.

$Rule_{op} ::= Rule_{op} \vee Rule_{op} \mid pathRuleExpr \mid Attexp$

$pathRuleExpr ::= pathRuleExpr \wedge pathRuleExpr \mid (pathLabelExpr)$

$pathLabelExpr ::= pathLabelExpr.pathLabelExpr \mid$ edgeExp

$Attexp$ ::= $Attexp \wedge Attexp \mid$ uexp = value $\mid$ vexp = value

$edgeExp ::= edgeExp \wedge edgeExp \mid$ edgeuexp = value $\mid$ edgevexp = value $\mid$ edgeattexp = value

where, value is a atomic constant.

$uexp \in \{va(u) | va \in VA\}$, u is a formal parameter.

$vexp \in \{va(v) | va \in VA\}$, v is a formal parameter.

$edgeuexp \in \{va(e.u) | va \in VA\}$, e.u is a formal parameter.

$edgevexp \in \{va(e.v) | va \in VA\}$, e.v is a formal parameter.

$edgeattexp \in \{ea(e) | ea \in EA\}$, e is a formal parameter.

Here "." is the concatenation operator. The length of a pathLabelExpr is given by the number of concatenation operators plus 1. A pathLabelExpr can be split at the point of each . operator into edgeExp, and numbered sequentially, starting from 1 to the length of the pathLabelExpr.

Based on the stated POL_{AAR}, the following defines an access control system:

Definition 38. Attribute aware ReBAC system

An Attribute aware ReBAC system is a tuple, $\langle ARG, POL_{AAR}, checkAccess_{AAR} \rangle$ where ARG and POL_{AAR} are as in Def. 35 and 37, respectively. For an access request (a, b), $checkAccess_{AAR}$(a:V, b:V) $\equiv Rule_{op}$(a:V, b:V) where $Rule_{op}$ is evaluated as follows:

Step 1:

a. for each Attexp in $Rule_{op}$, substitute the values va(a) for va(u) and va(b) for va(v), where $va \in VA$.

b. For a pathLabelExpr in $Rule_{op}$, substitute True iff i) there exists a simple path p from a to b in ARG such that $|p|$ = length of pathLabelExpr, and ii) each sth edgeExpr of the pathLabelExpr where $1 \leq s \leq$ length of pathLabelExpr, evaluates to True. To evaluate sth edgeExpr, substitute va(e.u), ea(e), and va(e.v) by the corresponding $va \in VA$, $ea \in EA$, and $va \in VA$ attribute value assignments from $(2 \times s - 1)$th, $(2 \times s)$th, and $(2 \times s + 1)$th tuples in $pathLabel_{att}(p)$, respectively.

Step 2:

Evaluate the resulting boolean expression.

User a is permitted to do operation op on object b if and only if $Rule_{op}(a, b)$ evaluates to True.

For example, given ARG in Fig. 5.1, and $Rule_{op}$ = (Gender(e.u)=Female $\wedge$ Profession(e.u)=Student

$\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v)=Male $\wedge$ Profession(e.v)=Student), $Rule_{op}(Alice, Ron)$ evaluates to True.

Although both ReBAC and ABAC are powerful, flexible and comparable [2] in expressing authorization policies, relying solely on one is often insufficient. An example case will be used in order to compare ABAC policy presented in [13] and ReBAC policy in [11] with the proposed AReBAC policy in Def. 37. Consider the ARG in Fig. 5.1 and Table 5.1. Each row of table 5.1 represents a case and an associated authorization state example.

1. Row 1 indicates that both AReBAC and ReBAC policies can express the authorization state (Alice, Bob) whereas only ABAC rules cannot. ABAC rule fails because Alice and Cathy have the same attribute value combination. The generated ReBAC and AReBAC rules are "F.F.F" and "(Relation-type(e)=F.Relation-type(e)=F.Relation-type(e)=F)", respectively.

2. Row 2 indicates that both ABAC and AReBAC policies can express the authorization state (Ron, Bob) whereas only ReBAC rules cannot. ReBAC rule fails because there is only one path labeled "F.F" from Ron to Bob which is satisfied by unauthorized pair, such as, (Alice, Cathy). The generated ABAC and AReBAC rule is the same, "Gender(u)=Male $\wedge$ Profession(u)=Student $\wedge$ Gender(v)=Male $\wedge$ Profession(v)=Officer".

3. The 3rd row, authorization state (Alice, Ron), cannot be expressed by both ABAC and ReBAC. ABAC rule fails because Alice and Cathy have the same attribute value combination. ReBAC rule fails because the only path label "F" is satisfied by other unauthorized pairs, such as (Alice, Bob). The AReBAC rule is "(Gender(e.u)=Female $\wedge$ Profession(e.u)=Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v)=Male $\wedge$ Profession(e.v)=Student)".

4. The 4th row, Auth = $\{$(Bob, Alice)$\}$ is not expressible by only ABAC (Since (Bob, Cathy) will be allowed), only ReBAC (since no path exists from Bob to Alice), and AReBAC ((Bob, Cathy) will be allowed and no path exists).

According to the used policy specification language, AReBAC is more expressive than ABAC [13] and ReBAC [11]. Additionally, it can be clearly observed that, if entity ids are allowed,

Table 5.1: Example data

ReBAC	ABAC	AReBAC	AUTH
Yes	No	Yes	{(Alice, Bob)}
No	Yes	Yes	{(Ron, Bob)}
No	No	Yes	{(Alice, Ron)}
No	No	No	{(Bob, Alice)}

AReBAC policy will never fail (such as [6]). However, imposing this condition conflicts with core principles of ABAC and ReBAC. Therefore, the AReBAC policy specification in this study checks whether the target access control system could be generated avoiding explicit use of unique entity id. Based on this motivation, ARREP problem will be defined in the next section.

5.3 Attribute-aware ReBAC RuleSet Existence Problem

This section defines the ARREP along with a feasibility detection algorithm and its associated proof of correctness and complexity analysis.

Definition 39. Attribute aware ReBAC RuleSet Existence Problem(ARREP)
Given an EAS and an ARG as in Def. 34 and 35, respectively, where V=U, does there exist a RuleSet as in Def. 37 so that the resulting Attribute aware ReBAC system satisfies:

$$(\forall u, v \in U)[checkAccess_{AAR}(u, v) \Leftrightarrow checkAccess_{EAS}(u, v)]$$

Such a RuleSet, if it exists, is said to be a suitable RuleSet, otherwise the problem is said to be infeasible.

The following subsection develops a ARREP solution algorithm.

5.3.1 ARREP Solution Algorithm

Algorithm 5.1 resolves the ARREP problem. Given an ARREP instance, it returns either feasible status and $Rule_{op}$, or infeasible status, incomplete $Rule_{op}$ and failed authorizations. Given any graph, the task finding all possible simple paths from a source vertex to a target vertex is well known, hence, details of function FindAllSimplePath() in Algorithm 5.1 are not provided (it can

be adapted from [11]). The overall complexity of computing all possible paths from a vertex to another in ARG is $O(|E|!)$ as it considers only simple paths.

Theorem 8. *The overall complexity of ARREP feasibility detection Algorithm 5.1 is* $O(|V|^4 \times (|E|!))$.

Proof. In Algorithm 5.2, overall complexity of Lines 1, 4, 2-3 and 5-6 are O($|U|$), O($|U|$), O($|AUTH|$), and O($|AUTH|$), respectively. Therefore, overall complexity of Algorithm 5.2 is O($|AUTH|$). The overall complexity of Algorithm 5.3 is O($|V|$) since the maximum number of edges allowed in a simple path of ARG is $|V|$-1. Combining all these, the computational complexity of Algorithm 5.1 as follows: Lines 4-7 of Algorithm 5.1 give $O(|AUTH|^2)$ complexity. According to the complexity of FindAllSimplePath() noted before, Lines 9 and 13, both give $O(|E|!)$ complexity. The overall complexity of Lines 14-15 is $O(|V|^2 \times (|E|!))$, and the set intersection in Line 16 takes $O(|E|!)$. Lines 17-21 can be ignored compared to others, therefore, the loop from Lines 8-21 takes overall $O(|V|^4 \times (|E|!))$ complexity as the loop may iterate $|AUTH| \leq |V|^2$ times. Hence, the worst case complexity of Algorithm 5.1 is $O(|V|^4 \times (|E|!))$. □

The correctness proof of Algorithm 5.1 is similar to the feasibility detection algorithm in [11], and is therefore omitted. Although overall complexity of feasibility detection algorithm in [11] and Algorithm 5.1 are same, however, the latter may have more or less computation time. If Algorithm 5.2 succeeds $\forall(a, b) \in AUTH$, only O($|AUTH|^2$) will be the real computational complexity, which is linear compared to the computed worst case complexity. The computational complexity significantly reduces even if Algorithm 5.2 succeeds for some $(a, b) \in AUTH$ since avoiding all possible path generation from a source vertex to target vertex in ARG (FindAllSimplePath() in Line 9) to any extent helps. Otherwise, taking both user and edge attribute value combination into consideration certainly adds overhead to the computation time of Algorithm 5.1, compared to feasibility detection algorithm in [11].

Let us consider the ARG in Fig. 5.1 where Range(Relation-type) is changed from $\{Friendship\}$ to $\{Friendship, Parent\}$. Since the “Parent" relation is not present anywhere as edge attribute

Algorithm 5.1 ARREP Solution Algorithm

Input: An EAS and an ARG where V=U.

Output: Feasible/infeasible status and $Rule_{op}$. If infeasible, set of failed authorization tuples.

1: $Rule_{op}$:= NULL
2: failedAuthPairs := $\emptyset$
3: tempAUTH := AUTH
4: **for** each $(a, b) \in tempAUTH$ **do**
5: **if** ABAC-Expr(EAS, VA, UATTValue, a, b) == SUCCESS **then**
6: **if** $Rule_{op}$ is NULL **then** $Rule_{op} := \bigwedge_{va \in VA} va(u) = va(a) \wedge \bigwedge_{va \in VA} va(v) = va(b)$ **else** $Rule_{op} := Rule_{op} \vee \bigwedge_{va \in VA} va(u) = va(a) \wedge \bigwedge_{va \in VA} va(v) = va(b)$
7: $tempAUTH \setminus := \{(a, b)\}$
8: **while** $\exists (a, b) \in tempAUTH$ **do**
9: $SP(a, b)$:= FindAllSimplePath(a,b, ARG)
10: **if** $SP(a, b)$ = $\emptyset$ **then**
11: $failedAuthPairs := failedAuthPairs \cup \{(a, b)\}$ //Not Feasible for (a,b) tuple
12: $tempAUTH \setminus := \{(a, b)\}$ and **Continue**
13: $PATHLABEL_{att}(a.b) := \{pathLabel_{att}(p) | p \in SP(a, b)\}$
14: **for** each $pl \in PATHLABEL_{att}(a.b)$ **do**
15: $SAT_{ab}(pl)$ = $\{(c, d) \in V \times V |$ there exists a simple path s from c to d in ARG, c$\neq$d, (c,d)$\notin$AUTH, pl=$pathLabel_{att}(s)\}$
16: Q_{ab} := $\bigcap_{pl \in PATHLABEL_{att}(a.b)} SAT_{ab}(pl)$
17: **if** $Q_{ab} \neq \emptyset$ **then**
18: $failedAuthPairs := failedAuthPairs \cup \{(a, b)\}$ //Not Feasible for (a,b) tuple
19: $tempAUTH \setminus := \{(a, b)\}$ and **Continue**
20: **if** $Rule_{op}$ is NULL **then** $Rule_{op} := \bigwedge_{pl \in PATHLABEL_{att}(a.b)} (generateRule(pl))$ **else** $Rule_{op} := Rule_{op} \vee \bigwedge_{pl \in PATHLABEL_{att}(a.b)} (generateRule(pl))$
21: $tempAUTH \setminus := \{(a, b)\}$
22: **if** failedAuthPairs is $\emptyset$ **then**
23: **return** "feasible" and $Rule_{op}$
24: **else**
25: **return** "infeasible" and failedAuthPairs and $Rule_{op}$

Algorithm 5.2 ABAC-Expr

Input: EAS, VA, UATTValue, vertex a, vertex b.
Output: SUCCESS or FAILURE

1: $R1 = \{u1 | \forall va \in VA.va(a) = va(u1)\}$
2: **if** $\exists u1, u2 \in R1.(u1, u3) \in Auth \wedge (u2, u3) \in \overline{Auth}$ where $u3 \in V$ **then**
3: **return** FAILURE
4: $R2 = \{u4 | (\forall va \in VA.va(b) = va(u4)\}$
5: **if** $\exists u4, u5 \in R2.(u4, u6) \in Auth \wedge (u5, u6) \in \overline{Auth}$ where $u6 \in V$ **then**
6: **return** FAILURE
7: **return** SUCCESS

Algorithm 5.3 generateRule

Input: String pathlabel
Output: String rule

1: rule := NULL
2: SubStr := splitStr(pathlabel,".") // The splitStr function splits pathlabel using . into an ordered list of substrings, and return the saved substrings into an array.
3: numEdges := (number of elements in SubStr-1)÷ 2
4: //rm function returns the given string after removal of leading "(" and trailing ")"
5: **for** i = 1 to numEdges **do**
6: tempu := splitStr(rm(SubStr[2*i-1]), ",")
7: tempv := splitStr(rm(SubStr[2*i+1]), ",")
8: tempe := splitStr(rm(SubStr[2*i]), ",")
9: **if** rule is NULL **then** rule := $\bigwedge_{1 \le j \le m} va_j(e.u) = tempu[j] \wedge va_j(e.v) = tempv[j] \wedge \bigwedge_{1 \le k \le n} ea_k(e) = tempe[k]$ **else** rule := rule . $\bigwedge_{1 \le j \le m} va_j(e.u) = tempu[j] \wedge va_j(e.v) = tempv[j] \wedge \bigwedge_{1 \le k \le n} ea_k(e) = tempe[k]$ //. means the concatenation
10: **return** rule

in the ARG, the effect of introducing a new user with “Parent" relation in ARG remains undetermined. This might happen to any ARG with a particular rule structure as change in relationships or adding a new user may effect the validity of the current rule set. We call this “unrepresented path labels" problem in ARG. The rule structure in this study compares direct values, the $Rule_{op}$ generated by Algorithm 5.1 does consider all user and edge attributes, and ARG is static by nature. Thereby, unrepresented path labels does not impact the $Rule_{op}$.

In order to show a comparison with our AReBAC policy language in user to user relationship context, the model presented in [18] is compared as follows:

- By construction, the policy language in this study does not support inverse relationship and count attribute as in [18].

- The policy language in Def. 37 is unable to count the number of existing paths between access initiator and target users. Another example is, the policy language in Def. 37 is unable to compare attribute value assignments of any two particular users along the path from initiator to target in ARG.

- The policy language in [18] supports the common regular expression feature, wildcard (* means to 0 to any number), optional (? means 0 or 1) notation, and negative path expression, while this study completely ignores them.

Clearly the AReBAC rule structure presented in this study is not the most general one. More expressiveness can be added such as in [18] and current feasibility problem statement could be correspondingly reformulated.

5.4 ARREP Infeasibility Solution

Given an infeasible ARREP instance as in Def. 39, an infeasibility solution basically generates a RuleSet which completes the AReBAC system. Formally,

1. An infeasibility solution is said to be exact iff:
   $(\forall u, v \in U)[checkAccess_{AAR}(u, v) \Leftrightarrow checkAccess_{EAS}(u, v)]$

2. An infeasibility solution is said to be approximate iff:
   $(\exists u, v \in U)[checkAccess_{AAR}(u, v) \neq checkAccess_{EAS}(u, v)]$

   An infeasibility solution is said to be inclusive approximate only if:
   $(\forall u, v \in U)[checkAccess_{AAR}(u, v) = True \rightarrow (u, v) \in AUTH]$

In this section, different approaches for ARREP infeasibility solution will be discussed briefly. Later on, algorithms for infeasibility solution will be presented.

5.4.1 Exact Solution

In this subsection, an exact solution to infeasibility in ARREP will be discussed with computational complexity as well as shortcomings. It is accomplished by adding edges to the given ARG as follows:

Definition 40. Add relationship edge

Given an ARREP infeasible instance, Algorithm 5.1 returns a set of failed authorization pairs, failedAuthPairs. Subsequently, the following steps are used:

1. It is assumed that, $\forall ea \in EA.op \notin Range(ea)$.
2. $\forall ea \in EA, Range(ea) \cup := op$, where $op \in OP$.
3. For each $(a, b) \in failedAuthPairs$, E := E $\cup$ {(a,b,op,op,...,op)}.
   Note*: for all the newly added edges, $\forall ea \in EA.ea(e) = op$.
4. $Rule_{op} := Rule_{op} \vee (\bigwedge_{ea \in EA} ea(e) = op)$, $Rule_{op}$ is returned by Algorithm 5.1.

For example, given the previous infeasible example where Auth = $\{(Bob, Alice)\}$ and ARG as in Fig. 5.1, Fig. 5.2 shows the additional edge from Bob to Alice, labeled by the operation op$\in OP$ where op is added to Range(Relation-type). The following theorem proofs the correctness of the stated infeasibility correction approach in Def. 40.

Theorem 9. *Def. 40 provides an exact solution to infeasibility in ARREP.*

Proof:

As stated, for all $(a, b) \in failedAuthPairs$, adding an edge from vertex a to b in ARG creates a path of length 1. Clearly, by the checkAccess evaluation presented in Def. 38, all $(a, b) \in failedAuthPairs$ satisfies ($\bigwedge_{ea \in EA} ea(e) = op$), and therefore, adding only one term is sufficient enough for a operation $op \in OP$. Since it is assumed that, $\forall ea \in EA.op \notin Range(ea)$, therefore, no other $U \times U \setminus failedAuthPairs$ satisfies ($\bigwedge_{ea \in EA} ea(e) = op$). Hence, the claim is correct.

As stated in Def. 40, the solution adds upto $|AUTH|$ edges to the ARG. Therefore, the worst

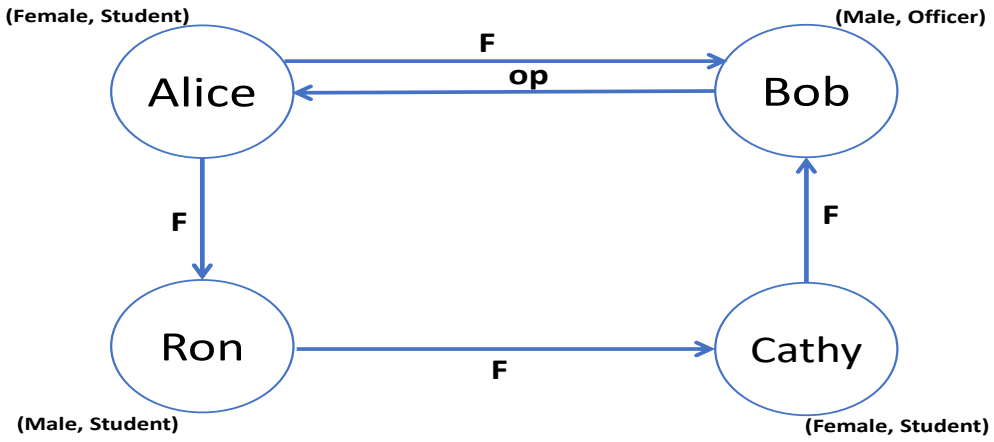

Figure 5.2: Additional Relationship edge from bob to Alice

case complexity is linear to the number of authorizations given. However, this solution approach has limitations. For example, less number of additional edges could be used to resolve the infeasibility ([11]). Furthermore, there might be cases where it is undesirable to alter the given ARG and Range of attributes at all. Keeping this conditions in mind, the next subsection presents an inclusive approximate infeasibility solution approach by using additional information to the rule only, enclosed by [].

5.4.2 Inclusive Approximate Solution

Given an ARG, a radiusValue$\in Z^+$, and a user $a \in U$, Algorithm 5.4 returns the set of neighbors within distance radiusValue$\in Z^+$, denoted by Neighbor[a]. Algorithm 5.4 uses the basic principles of Breadth First Search (BFS) graph traversal algorithm. Given ARG in Fig. 5.3, user u_0, and radiusValue 2, Algorithm 5.4 returns $\{(U_{12}, 1), (U_{13}, 1), (U_{21}, 1)\}$.

It can be observed that given a starting vertex, Algorithm 5.4 considers the direction of edges while searching for neighbors. A slight modification to Algorithm 5.4, for example, making the search undirected, could change the number of neighbors found.

Given the same example: ARG in Fig. 5.3, user u_0, and radiusValue 2, modifying Algorithm

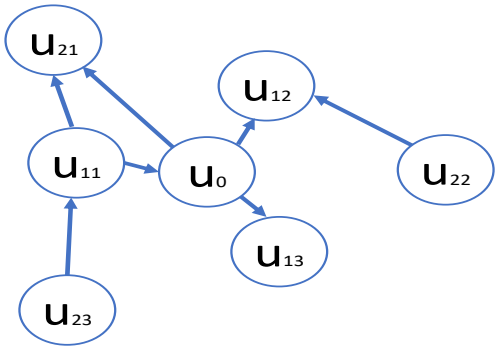

Figure 5.3: Example ARG (user and edge attribute information are ignored).

5.4 with undirected search returns the following:
$\{(U_{11}, 1), (U_{12}, 1), (U_{13}, 1), (U_{21}, 1), (U_{22}, 2), (U_{23}, 2)\}$. We leave it to the system designer to utilize the neighbor search direction.

It is evident that the complexity of Algorithm 5.4 is linear: O($|E| \times RadiusValue$). In this study, the radiousValue will be kept to 1. The following definition specifies the additional information to resolve infeasibility:

Definition 41. For any $(a, b) \in U \times U, u \neq v$, the additional information is denoted by $Neighbor(a, b)$ defined as follows:

$\bigwedge_{va \in VA, (c,i) \in Neighbour(a)} va(u, i) = va(c) \wedge \bigwedge_{va \in VA, (d,j) \in Neighbour(b)} va(v, j) = va(d)$ where va(u,i) and va(v,j) are formal parameters, va(c) and va(d) are va attribute value assignment of actual user c and d.

Based on the defined additional information regarding any pair of vertices in ARG, the following definition lists the steps to resolve infeasibility:

Algorithm 5.4 Approximate Infeasibility Solution Algorithm

Input: ARG, user a, RadiusValue$\in Z^+$
Output: Neighbor[a]

1: $\forall u \in U, USED[u] := FALSE$
2: USED[a] := True
3: Neighbor[a] := $\emptyset$
4: Prev := $\{a\}$
5: **for** r = 1 to RadiusValue **do**
6: newPrev := $\emptyset$
7: //Given an edge e1$\in E$, e1.u1 and e1.v1 denotes the starting and terminating vertices, respectively
8: **for** $e1 \in E$ **do**
9: **if** $e1.u1 \in Prev \wedge USED[e1.v1] == FALSE$ **then**
10: Neighbor[a] := Neighbor[a] $\cup\{(e1.v1, r)\}$
11: USED[e1.v1] := TRUE
12: newPrev $\cup :=$ $\{e1.v1\}$
13: **if** newPrev == $\emptyset$ **then**
14: **return** Neighbor[a]
15: Prev := newPrev
16: **return** Neighbor[a]

Definition 42. For each $(a, b) \in failedAuthPairs$, $Rule_{op} := Rule_{op} \vee \bigwedge_{va \in VA} va(u) = va(a) \wedge \bigwedge_{va \in VA} va(v) = va(b) \wedge [Neighbour(a, b)]$ where the following assumption holds:

there does not exist a $(f, g) \in \overline{AUTH}$ such that i) Neighbour(a,b) = Neighbour(f,g), and ii) the conjunctive term $\bigwedge_{va \in VA} va(u) = va(a) \wedge \bigwedge_{va \in VA} va(v) = va(b)$ turns true by substituting the value of va(u) and va(v) by va(f) and va(g), respectively.

Note*: The additional information is enclosed in [] is not a part of the AReBAC rule structure, therefore, it is evaluated separately from the rule. The $Rule_{op}$ will be evaluated as stated in Def. 38. For any pair $(s, t) \in U \times U, s \neq t$ the additional information enclosed in [] is substituted by True only if Neighbor(s,t) = additional information.

Based on the additional information above, the following theorem proves that the proposed infeasibility solution is inclusive approximate one:

Theorem 10. *As stated in Def. 42, the proposed infeasibility solution is inclusive approximate.*

Proof

According to the assumption provided in Def. 42, it is trivial.

More approaches of infeasibility correction are left as future works.

5.5 Use Cases and Implementation

In this section, use cases with significant results will be discussed briefly. Later on, some implementation details will be listed with rule minimization technique.

5.5.1 Infeasibility solution with additional information

Given ARG as in Fig. 5.1, AUTH = $\{(Bob, Alice)\}$, and radiusValue = 1, Neighbor[Bob] and Neighbor[Alice] computed by Algorithm 5.4 is given by $\emptyset$ and $\{(Ron, 1), (Bob, 1)\}$, respectively. Therefore, by Def. 42:

$$Rule_{op} = gender(u) = Male \wedge Profession(u) = Officer \wedge gender(v) =$$
$$Female \wedge Profession(v) = Student \wedge [gender(v, 1) = Male \wedge Profession(v, 1) =$$
$$Student \wedge gender(v, 1) = Male \wedge Profession(v, 1) = Officer]$$

As noted before, utilizing the neighbor search direction has it's pros and cons. If undirected search for neighbors is used, the example of AUTH = $\{(Bob, Alice)\}$ can not be resolved as Alice and Cathy will have the same set of neighbors. Since directed search as stated in Algorithm 5.4 is used, Neighbor[Cathy] = $\{(Bob, 1)\}$. However, undirected neighbor search increases the scope of finding neighbors within the same computational complexity. We leave further analysis on this as future work.

5.5.2 Order of operations in Algorithm 5.1

Let Auth be a singleton. It is evident that, for a pair $(a, b) \in AUTH$, Algorithm 5.1 checks whether it is possible to generate the conjunctive term with the user attribute value assignments only. If that step fails, it proceeds with all possible path generation from vertex a to b in ARG and follows the

rest of the procedure. For example, given ARG as in Fig. 5.1 and Auth ={(Ron, Bob)}, considering Attexp first to generate the conjunctive term results in $\langle Gender(u) = Male \wedge Profession(u) = Student \wedge Gender(v) = Male \wedge Profession(v) = Officer \rangle$ whereas the reverse order generates (Gender(e.u) = Male $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Female $\wedge$ Profession(e.v) = Student . Gender(e.u) = Female $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Male $\wedge$ Profession(e.v) = Officer). By observation, it can be precisely concluded, the $Rule_{op}$ might change due to the order but feasibility result does not change. However, considering the user attribute information first can significantly reduce the computational complexity if it succeeds, remains same otherwise.

5.5.3 Implementation

Although the rule structure defined in Def. 37 allows any possible combination of user and edge attributes, Algorithm 5.1 does not eliminate any user or edge attribute while generating $Rule_{op}$. Attribute elimination may results in unrepresented partition and path labels problem, therefore, rule minimization in this study is limited to finding minimal number of path labels in conjunctive terms only. As stated in Algorithm 1, for a tuple (a,b) in AUTH, the conjunctive term is formed by AND'ing all possible path labels from a to b iff i) Algorithm 5.2 fails, and ii) the conjunctive term evaluates false for all unauthorized tuples. Instead of using all possible path labels in the conjunctive term of such (a,b), the smallest possible subset (except empty set) of those is used to form the conjunctive term, ensuring that the minimal size conjunctive evaluates false for all unauthorized tuples. For instance, given ARG in Fig. 5.1 and AUTH = $\{(Alice, Bob)\}$, i) Algorithm 5.2 returns FAILURE for (Alice, Bob), ii) there exists two paths, say p1 and p2, from Alice to Bob in ARG where $pathLabel_{att}(p1)$ and $pathLabel_{att}(p2)$ are (Female, Student).(F).(Male,Officer) and (Female, Student).(F).(Male,Student).(F).(Female, Student).(F).(Male,Officer). Without any rule minimization, $Rule_{op}$ generated by Algorithm 5.1 is given by the conjunction of generateRule($pathLabel_{att}(p1)$) and generateRule($pathLabel_{att}(p2)$), given by, (Gender(e.u) = Female $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v)

= Male $\wedge$ Profession(e.v) = Officer) $\wedge$ (Gender(e.u) = Female $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Male $\wedge$ Profession(e.v) = Student . Gender(e.u) = Male $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Female $\wedge$ Profession(e.v) = Student . Gender(e.u) = Female $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Male $\wedge$ Profession(e.v) = Officer). The possible subset of path labels in this case is: either one or both. It is evident that, i) only $pathLabel_{att}(p1)$ not possible because it is satisfied by unauthorized pair (Cathy, Bob) ii) only $pathLabel_{att}(p2)$ is possible because it is not satisfied by any unauthorized pair. Thereby, minimum subset of path labels in the conjunction principal reduces $Rule_{op}$ to (Gender(e.u) = Female $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Male $\wedge$ Profession(e.v) = Student . Gender(e.u) = Male $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Female $\wedge$ Profession(e.v) = Student . Gender(e.u) = Female $\wedge$ Profession(e.u) = Student $\wedge$ Relation-type(e) = F $\wedge$ Gender(e.v) = Male $\wedge$ Profession(e.v) = Officer). A simple Java implementation of ARREP with sample randomly generated cases where the specified rule minimization technique has been utilized is available upon request.

CHAPTER 6: EXTENDING THE FEASIBILITY OF RELATIONSHIP AND ATTRIBUTE-BASED ACCESS CONTROL POLICY MINING

Access control policy mining problem seeks to automate the process of obtaining an equivalent target policy when a complete source access control system along with supporting data is given. Access control policy mining algorithms offer promising advancement in automating policy generation, whereas manual effort requires more time, and could be error-prone.

Feasibility analysis of ReBAC policy mining investigates whether migration to a ReBAC system is possible or not when a complete access control system along with supporting relationship data is given. This chapter studies feasibility analysis of ReBAC policy mining from a different angle, formally named as Extended ReBAC RuleSet Existence Problem (ERREP). Various directions of ERREP are discussed, defined, and solved algorithmically, along with complexity analysis. A similar study has been conducted on feasibility analysis of ABAC policy mining as well, Extended ABAC RuleSet Existence Problem (EAREP) and variations are defined and solved precisely. In both ReBAC and ABAC cases, notions of exact and approximate infeasibility solutions are developed and studied with example cases.

6.1 Motivation

Before further discussion, the exact and approximate terms should be cleared briefly. Given a source and a target access control system, both are equivalent/exact only if i) they have identical user and resource/object sets, and ii) the authorizations (who can access what) are same, approximate otherwise.

Consider a scenario where a complete ReBAC system is given. It is trivial to generate the equivalent enumerated set of authorizations from the given ReBAC system. If some authorizations are removed or added from the generated set of authorizations, it is no longer equivalent to the given ReBAC policy. For example, consider the EAS and ReBAC policy definition in [11]. The directed RG in Fig. 6.1 where V={Alice, Bob, Ray,Tom}, E={(Alice, Tom, F), (Tom, Ray, F),

(Tom, Bob, F), (Bob, Ray, F)}, $\Sigma = \{F\}$ and $Rule_{op} = F.F$. The equivalent set of authorizations is clearly, AUTH = { (Alice, Ray), (Alice, Bob), (Tom, Ray)} and let the EAS tuple consists of identical user set V, OP = $\{op\}$, and AUTH relation. Now, let two authorizations are removed from AUTH, and it becomes AUTH = { (Alice, Ray)}. Now the given ReBAC policy is representing more authorizations that the modified AUTH relations.

To generate an equivalent ReBAC policy of the modified authorization set, starting from scratch with an ReBAC policy mining algorithm (Such as [11]) is an obvious option. However, the question is, can we use the existing ReBAC policy somehow to generate the target ReBAC policy? We formulate this question by defining a new problem, Extended ReBAC RuleSet Existence Problem (ERREP). A few varieties of ERREP are introduced briefly as well.

Similar objectives draw the attention to Extended ABAC RuleSet Existence Problem (EAREP). If some authorizations are removed or added from the given ABAC system, it is no longer remain equivalent. Therefore, the similar question raises, can we use the existing ABAC policy somehow to generate the target ABAC policy?

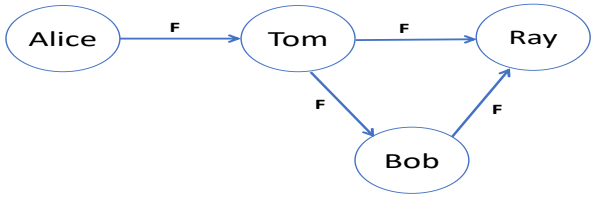

Figure 6.1: Directed RG example

6.2 Extended ReBAC RuleSet Existence Problem

In this section, some preliminaries of Extended ReBAC RuleSet Existence Problem (ERREP) will be noted first. Later, the solution and infeasibility will be discussed briefly.

6.2.1 Source and Target Access Control System

This chapter studies Extended ReBAC RuleSet Existence problem with EAS (Enumerated Authorization System) and ReBAC system as source access control systems. As mentioned earlier, the ERREP problem looks for an ReBAC policy, therefore, the target access control system is ReBAC system as well. Although the idea of ERREP can be accommodated with any ReBAC model, in order to keep continuity, the EAS and ReBAC system defined in chapter 4 are adopted without alteration.

Before defining ERREP and variations, a precise measurement of rule quality is needed to be established since the success of ERREP depends on the amount of reuse of the existing ReBAC policy. The rule quality metric is defined as follows:

Definition 43. Rule Quality Metric for ERREP

Given a $Rule_{op}$ for an $op \in OP$ using the Def. 28, let the number of pathRuleExpr in $Rule_{op}$ is denoted by $Card(Rule_{op})$. Let $Rule_{op}$ does not have any duplicate pathRuleExpr and pr denotes a pathRuleExpr in $Rule_{op}$ which is decomposed into pathLabelExprs. The set of pathLabelExprs present in pr is denoted by pLabels(pr). Let $Rule'_{op}$ denotes another rule for the same $op \in OP$ which follows the same rule structure as $Rule_{op}$, and $Rule'_{op}$ does not have any duplicate pathRuleExpr. Given $Rule_{op}$ and $Rule'_{op}$ where $op \in OP$ using the Def. 28,

commonTerms($Rule_{op}$, $Rule'_{op}$) = $\{pr' | pr'$ is a pathRuleExpr in $Rule'_{op}$ where there exists a pathRuleExpr pr in $Rule_{op}$ such that pLabels(pr') $\subseteq$ pLabels(pr) $\}$.

The rule quality metric of a $Rule_{op}$ w.r.t. $Rule'_{op}$ is given by a non negative numeric value:

$|commonTerms(Rule_{op}, Rule'_{op})|/Card(Rule'_{op})$.

Clearly, the maximum possible value of rule quality metric is 1.

Given $Rule_{op}$ and $Rule'_{op}$ as $F.F \wedge F.F.F$ and $F.F$, respectively, commonTerms($Rule_{op}$, $Rule'_{op}$) = $\{F.F\}$ since $\{F.F\} \subseteq \{F.F.F, F.F\}$. Subsequently, rule quality metric of $Rule_{op}$ w.r.t. $Rule'_{op}$ is given by 1.

Based on the foregoing, Extended ReBAC RuleSet Existence Problem (ERREP) is defined as

follows:

Definition 44. Extended ReBAC RuleSet Existence Problem (ERREP)

Given $\langle$U,AUTH,$checkAccess_{EAS}\rangle$ as an EAS and $\langle RG, POL_{ReBAC}, checkAccess_{ReBAC}\rangle$ as a ReBAC system, as shown in Def. 24 and 29, respectively, where i) RuleSet consists of a rule $Rule'_{op}$, ii) V=U, and iii) AUTH $\neq \{(u,v)|u,v \in U, checkAccess_{ReBAC}(u,v) = true\}$, does there exist a $Rule_{op}$ as defined in Def. 28 such that $Rule'_{op}$ is substituted in the given ReBAC system by $Rule_{op}$ and the resulting ReBAC system satisfies:

- $(\forall u, v \in U)[checkAccess_{ReBAC}(u,v) \Leftrightarrow checkAccess_{EAS}(u,v)]$
- Rule quality metric as in Def. 43, $|commonTerms(Rule_{op}, Rule'_{op})|/|Rule'_{op}|$ is maximized.

Such a RuleSet consisting of $Rule_{op}$, if it exists, is said to be a suitable RuleSet, otherwise the problem is said to be infeasible.

For example, consider the given RG in Fig. 6.1, $Rule'_{op} = F.F$, and AUTH = { (Alice, Ray)}. Clearly, $Rule'_{op}$ satisfies { (Alice, Ray), (Alice, Bob), (Tom, Ray)}. Substituting $Rule'_{op}$ by $Rule_{op} = F.F \wedge F.F.F$ makes the EAS and ReBAC system equivalent and the rule quality metric is 1.

Based the comparison between AUTH and $\{(u,v)|u,v \in U, checkAccess_{ReBAC}(u,v) = true\}$, ERREP is classified into three types as follows:

Definition 45. ERREP Variations

Classification of ERREP completely follows the ERREP defined as in Def. 44 only except condition (iii) which is substituted by the following accordingly:

1. ERREP-0: AUTH$\subset \{(u,v)|u,v \in U, checkAccess_{ReBAC}(u,v) = true\}$
2. ERREP-1: $\{(u,v)|u,v \in U, checkAccess_{ReBAC}(u,v) = true\} \subset AUTH$
3. ERREP-2: AUTH$\cap\{(u,v)|u,v \in U, checkAccess_{ReBAC}(u,v) = true\} \neq \emptyset$ (assuming it is not ERREP 0 or1)

This simple variations make subtle differences in the solving technique of ERREP. The subsequent discussion will carry out solutions to all variations of ERREP 0-2.

Theorem 11. *ERREP-1 can be solved by using Algorithm 4.1.*

Proof

Trivial. By definition of ERREP-1, ReBAC rule is required to be found for only the authorization set, $AUTH \setminus \{(u,v)|u,v \in U, checkAccess_{ReBAC}(u,v) = true\}$ as the rest of the authorizations are already covered by given $Rule'_{op}$. The parameters to be sent to Algorithm 4.1 are fairly straightforward, and the resulting rule is OR'ed with the given $Rule'_{op}$. Even in the the case of infeasibility, an exact infeasibility solution is provided in Chapter 4. Therefore, exact solution is guaranteed.

6.2.2 ERREP-0 Algorithm

In this subsection, the solution algorithm 6.1 for ERREP-0 is presented along with associated proofs and complexity analysis. This algorithm iterates through each pathRuleExprs in the given $Rule'_{op}$, and looks for a suitable $Rule_{op}$ which substitutes $Rule'_{op}$ in the given ReBAC system to make it equivalent to the given EAS, and ensures the maximum utilization of $Rule'_{op}$. If no such $Rule_{op}$ is found, it looks into the Algorithm 4.1 to generate rule for the authorizations left out. If the Algorithm 4.1 returns infeasible as well, Algorithm 6.1 returns infeasible status.
Let PATHLABEL denotes the set of pathlabels for any vertex pair in the RG. Formally, PATHLABEL(c,d) := $\{pathLabel(p)|\ c,d \in V, c \neq d$ and p is a simple path from c to d in RG$\}$.

Theorem 12. *Asymptotic complexity of ERREP-0 Algorithm 6.1 is* $O((Card(Rule'_{op}) \times 2^{|V|^2}) + |V|^4 \times |E|!)$

Proof:

For simplicity, we assume for each pair (a,b) in $V \times V, a \neq b$, PATHLABEL(a,b) is precomputed using $O(|V|^2 \times |E|)$ complexity [11]. We also assume, subsets in line 11 are accessed in decreasing order.

Algorithm 6.1 ERREP-0 Solution Algorithm

Input: An ERREP-0 problem instance.

Output: Feasible/infeasible status and $Rule_{op}$.

1: $Rule_{op}$:= NULL, failedPathRuleExpr := $\emptyset$
2: **for** each pathRuleExpr pr in $Rule'_{op}$ **do**
3: $SAT_{pr} := U \times U$
4: **for** each pl in pLabels(pr) **do**
5: SAT_{pl} = $\{(c,d) \in V \times V|$ there exists a simple path s from c to d in RG, c≠d, pl=$pathLabel(s)\}$
6: $SAT_{pr} \cap := SAT_{pl}$
7: $unAuth_{pr}$:= $SAT_{pr} \setminus AUTH$
8: $Auth_{pr}$:= $SAT_{pr} \setminus unAuth_{pr}$
9: **if** $Auth_{pr}$ is $\emptyset$ **then** Continue
10: **if** $unAuth_{pr}$ is $\emptyset$ **then** $Rule_{op} := Rule_{op}. \vee pr$; Continue
11: **for** each $subset \in 2^{Auth_{pr}} \setminus \emptyset$ **do**
12: PL := $\bigcap_{(a,b) \in subset} PATHLABEL(a,b)$
13: **if** $\nexists (c,d) \in unAuth_{pr}.PL \subseteq$ PATHLABEL(c,d) **then** Break
14: **if** PL is $\emptyset$ **then** $failedPathRuleExpr \cup := pr$; Continue
15: **if** $Rule_{op}$ is Null **then** $Rule_{op} := \bigwedge_{pl \in PL \cup pLabels(pr)} pl$ **else** $Rule_{op} := Rule_{op}. \vee \bigwedge_{pl \in PL \cup pLabels(pr)} pl$
16: $Rule'_{op}$ substitutes $Rule_{op}$
17: $AUTH_{ReBAC} = \{(u,v)|u,v \in V \wedge checkAccess_{ReBAC}(u,v) = True\}$.
18: **if** $AUTH_{ReBAC} == AUTH$ **then return** "Feasible" and $Rule_{op}$
19: $\langle Status, Rule''_{op}, failedAuthPairs \rangle$:= RREP-0(EAS, RG) //Calling Algorithm 4.1 where line 3 is substituted by AUTHset := $AUTH \setminus AUTH_{ReBAC}$
20: **if** Status == "feasible" **then return** "Feasible" and $Rule_{op}.\vee := Rule''_{op}$ **else**
21: **return** "Infeasible" and $Rule_{op}.\vee := Rule''_{op}$ and failedAuthPairs

In order to compute Algorithm 6.1 complexity, the for loop from line 2-15 is the dominant factor. The for loop from line 2-15 runs $Card(Rule'_{op})$ times. The for loop from line 4-6 takes $O(|V|^2 \times (|V| + |E|))$ complexity at the worst case. In addition to this, the for loop of line 11-13 takes $O(2^{|V|^2})$.

Therefore, overall complexity of the loop from line 2-15 is $O(Card(Rule'_{op}) \times 2^{|V|^2})$. Besides, Algorithm 4.1 in line 19 takes $O(|V|^4 \times |E|!)$ complexity. Therefore, overall complexity of ERREP-0 feasibility detection Algorithm 6.1 is $O((Card(Rule'_{op}) \times 2^{|V|^2}) + |V|^4 \times |E|!)$, considering the rest of the lines produce significantly less complexities.

Based on the complexity analysis, the ERREP-0 algorithm has additional complexity compared to the starting from scratch using Algorithm 4.1. However, there are multiple significant factors that should be counted for performance analysis because i) there might be cases when ERREP-0 algorithm is able to completely avoid the use Algorithm 4.1, and ii) line 10 in ERREP-0 algorithm succeeds frequently means the overall complexity can get significantly lower. Therefore, ERREP-0 algorithm can provide much better performance based on case to case analysis and we leave the rest of the analysis for the future work.

For example, consider the directed RG in Fig. 6.1 where V=$\{$Alice, Bob, Ray,Tom$\}$, E=$\{$(Alice, Tom, F), (Tom, Ray, F), (Tom, Bob, F), (Bob, Ray, F)$\}$, $\Sigma = \{F\}$ and $Rule_{op} = F.F$. The equivalent set of authorizations is clearly, AUTH = $\{$ (Alice, Ray), (Alice, Bob), (Tom, Ray)$\}$ and let the EAS tuple consists of identical user set V, OP = $\{op\}$, and AUTH = $\{$ (Alice, Ray)$\}$. Clearly, it is an ERREP-0 instance. By using Algorithm 6.1, $Rule_{op} = F.F \wedge F.F.F$ is generated. The generated $Rule_{op}$ substitutes $Rule'_{op}$ in the given ReBAC system, and makes the EAS and ReBAC system equivalent.

Theorem 13. *Algorithm 6.1 is correct.*

Proof:

Algorithm 6.1 is correct only if the following statement holds:

Given a ERREP-0 instance as in Def. 44, a suitable RuleSet exists iff Algorithm 6.1 generates the

$Rule_{op}$.

Assume, Algorithm 6.1 generates the $Rule_{op}$. By specification, Algorithm 6.1 considers each individual pathRuleExpr in the given $Rule'_{op}$ and checks for the set of authorized tuples by them. If a pathRuleExpr generates all authorized tuples belongs to given AUTH, it is kept unaltered. On the other hand, if a pathRuleExpr generates all unauthorized tuples compared to the given AUTH, it is completely discarded. There is a point to be mentioned here. By the definition of commonTerms in Def. 43, rule quality metric counts only when the pathRuleExpr is used as it is or some pathlabels are AND'ed with it. It is trivial that, if the pathlabels are AND'ed with an individual pathRuleExpr, the set of tuples authorized by pathRuleExpr beforehand, either decreases or remains the same. We understand that removing some pathlabels from the pathRuleExpr is another way of utilizing it, however, during this study, this concept is ignored because we limit the concept to utilize the given rule only. If the pathRuleExpr does contain both authorized and unauthorized tuples, Algorithm 6.1 looks for the set of pathlabels to be AND'ed to the pathRuleExpr being considered so that all unauthorized tuples authorized by unaltered pathRuleExpr beforehand gets denied. Algorithm 6.1 continues the same step for each pathRuleExpr and at the end of iteration if it generated $Rule_{op}$ is not equivalent to the given AUTH, then it calls for Algorithm 4.1 to generate rest of the part of the $Rule_{op}$. It is already proved that Algorithm 4.1 is correct [11], therefore, by construction, if a suitable $Rule_{op}$ exists, algorithm 6.1 finds the $Rule_{op}$.

On the other hand, assume, a suitable $Rule_{op}$ as defined by Def. 44 and ERREP-0 exists. By construction, Algorithm 6.1 considers each individual pathRuleExpr in the given $Rule'_{op}$ and keeps, ignores and alters them based on the generated authorizations. If a pathRuleExpr generates all authorized tuples belongs to given AUTH, it is kept unaltered. On the other hand, if a pathRuleExpr generates all unauthorized tuples compared to the given AUTH, it is completely discarded. If the pathRuleExpr does contain both authorized and unauthorized tuples, Algorithm 6.1 looks for the set of pathlabels to be AND'ed by considering all possible paths for all authorized tuples satisfied by the pathRuleExpr and AUTH. By the definition of rule quality metric, Algorithm 5.1 considers every pathRuleExpr to maximize the goal, and does not use entity ids. If algorithm 6.1 fails, by

construction, Algorithm 4.1 all possible pathlabels to generate the rest of the rule [11]. Thereby, if a suitable $Rule_{op}$ exists, Algorithm 6.1 finds the $Rule_{op}$.
Hence, Algorithm 6.1 is correct.

Theorem 14. *ERREP-2 can be resolved by slightly modifying and combining the procedure of Algorithm 4.1 and 6.1.*

Proof

By investigation, trivial.

6.2.3 ERREP-0 Infeasibility Solution

This section discusses about an exact and an approximate infeasibility solution of the ERREP-0 problem.

6.2.4 Exact Solution

If Algorithm 6.1 fails, the exact solution is similar to the infeasibility solution defined in [11].

1. For each tuple in the failedAuthPairs, add a relationship edge labeled as op in the RG.
2. Call Algorithm 4.1 again.

Lemma 11. *The proposed Exact solution to ERREP-0 is correct.*

Proof

Trivial from [11].

6.2.5 Approximate Solution

Exact solution could be the most desired one, however, approximate solution proposed here will provide flexibility. The proposed solution in subsection 6.2.4 alters the given RG which might not be desirable in some cases. Therefore, an alternate approximate solution has been proposed here

which utilizes the aim of rule quality metric.

First step towards the approximate infeasibility correction is, add an attribute named "priority" to each vertex in the RG. The $Rule_{op}$ given in Def. 28 is modified in order to accommodate the use of priority value as follows:

$Rule_{op} ::= Rule_{op} \vee Rule_{op} \mid (pathRuleExpr, priorityOrder)$

$pathRuleExpr ::= pathRuleExpr \wedge pathRuleExpr \mid pathLabelExpr$

$priorityOrder ::= > \mid \phi$

$pathLabelExpr ::= pathLabelExpr.pathLabelExpr \mid edgeLabel$

$edgeLabel ::= \sigma, \sigma \in \Sigma$

where > and ϕ represent increasing and don't care orders, respectively, and $Rule_{op}$ consists of disjunction of (pathRuleExpr, priorityOrder) pairs.

The evaluation procedure of $checkAccess_{ReBAC}$(a:V, b:V) in a ReBAC system with the specified $Rule_{op}$ is as follows:

(i) for each (pathRuleExpr, Order) pair in $Rule_{op}$ substitute True if i) for each pathLabelExpr in the pathRulelExpr, there exists a simple path p from a to b in RG with path label pathLabelExpr, and ii) priority(a) and priority(b), denoting the vertex priority values associated with user a and b, respectively, follows the priorityOrder order, otherwise substitute False, (ii) evaluate the resulting boolean expression.

Note that, priorityOrder don't care resembles that the order does not matter.

Definition 46. Approximate Solution to ERREP-0

Approximate solution to infeasibility in ERREP-0 can be generated using the following steps.

1. Execute Algorithm 6.1: collect failedPathRuleExpr and $Rule_{op}$. Note that, calling Algorithm 4.1 can be omitted to reduce complexity.

2. $AuthPair := \emptyset$, $unAuthPair := \emptyset$

3. For each pathRuleExpr pr in failedPathRuleExpr, i) compute $Auth_{pr}$ and $AuthPair \cup := Auth_{pr}$, ii) compute $unAuth_{pr}$ and $unAuthPair \cup := unAuth_{pr}$. (Computation of $Auth_{pr}$

and $unAuth_{pr}$ is as shown in Algorithm 6.1).

4. Constitute a graph G' = (V', E') where the set of vertices V' := V, the set of edges E := $\{(c,d)|(c,d) \in unAuthPair$ or $(d,c) \in AuthPair\}$.

5. Run topological sort algorithm on G'. If there is an order, the solution is exact. If an order in found, assign priority values to the vertices in decreasing order. The final $Rule_{op}$ is $Rule_{op} := Rule_{op}. \bigvee_{pr \in failedPathRuleExpr} (pr, >)$. Otherwise, the solution is approximate and return the existing $Rule_{op}$.

Theorem 15. *Def. 46 is correct and complexity is linear.*

Proof

By investigation of the topological sort algorithm, correctness of the proposed steps are trivial. The worst complexity is linear as the complexity of running topological sort on a graph is linear.

6.3 Extended ABAC RuleSet Existence Problem

Similar to the previous section, this section defines some preliminaries of Extended ABAC RuleSet Existence Problem (EAREP). The solution to EAREP and remedies to infeasibility are also noted.

6.3.1 Source and Target Access Control System

This chapter studies Extended ABAC RuleSet Existence problem with EAS (Enumerated Authorization System) and ABAC system as source access control system. Similar to ERREP mentioned earlier, the EAREP problem looks for an ABAC policy, therefore, the target access control system is an ABAC system as well. Although the idea of EAREP can be accommodated with any ABAC model, in order to keep continuity, the EAS and ABAC system defined in chapter 3 are adopted without alteration.

Before defining EAREP and variations, a precise measurement of rule quality is needed to be established since the success of EAREP depends on the amount of reuse of the existing ABAC policy. The rule quality metric is defined as follows:

Definition 47. Rule Quality Metric for EAREP

Given a $Rule_{op}$ for an $op \in OP$ using the Def. 4, let the number of Atomicexp in $Rule_{op}$ is denoted by $Card(Rule_{op})$. Let $Rule_{op}$ does not have any duplicate Atomicexp and ar denotes an Atomicexp in $Rule_{op}$ which is decomposed into uexp and oexp. Let, uoLabel(ar) = {(ua,value1)|ua(u)=value1 is a uexp in ar}∪{(oa,value2)|oa(o)=value2 is a oexp in ar}. Let $Rule'_{op}$ denotes another rule for the same $op \in OP$ which follows the same rule structure as $Rule_{op}$, and $Rule'_{op}$ does not have any duplicate Atomicexp. Given $Rule_{op}$ and $Rule'_{op}$ where $op \in OP$ using the Def. 4, commonTerms($Rule_{op}$, $Rule'_{op}$) = $\{ar'|ar'$ is an Atomicexpr in $Rule'_{op}$ where there exists an Atomicexp ar in $Rule_{op}$ such that uoLabels(ar') $\subseteq$ uoLabels(ar) $\}$.

The rule quality metric of a $Rule_{op}$ w.r.t. $Rule'_{op}$ is given by a non negative numeric value: $|commonTerms(Rule_{op}, Rule'_{op})|/Card(Rule'_{op})$.

Clearly, the maximum possible value of rule quality metric is 1.

It is assumed that, set of operations OP is a singleton, given by $\{op\}$, thus eliminated from further definitions. Based on the foregoing, Extended ABAC RuleSet Existence Problem (ERREP) is defined as follows:

Definition 48. Extended ABAC RuleSet Existence Problem (EAREP)

Given an EAS = ⟨U,O, AUTH,$checkAccess_{AUTH}$⟩ and an ABAC system ⟨U, O, UAValue, OAValue, POL_{ABAC}, $checkAccess_{ABAC}$⟩ as in Def. 2 and 5, respectively, where i) RuleSet consists of a rule $Rule'_{op}$, ii) EAS and ABAC system have identical user and object sets, and iii) AUTH $\neq \{(u,o)|(u,o) \in U \times O, checkAccess_{ABAC}(u,v) = true\}$, does there exist a $Rule_{op}$ as defined in Def. 4 such that $Rule'_{op}$ is substituted in the given ABAC system by $Rule_{op}$ and the resulting ABAC system satisfies:

- $(\forall u, o \in U \times O)[checkAccess_{ABAC}(u,o) \Leftrightarrow checkAccess_{AUTH}(u,o)]$
- Rule quality metric as in Def. 47, $|commonTerms(Rule_{op}, Rule'_{op})|/|Rule'_{op}|$ is maximized.

Such a RuleSet consisting of $Rule_{op}$, if it exists, is said to be a suitable RuleSet, otherwise the problem is said to be infeasible.

Based the comparison between AUTH and $\{(u,o)|(u,0) \in U \times O, checkAccess_{ABAC}(u,o) = true\}$, EAREP is classified into three types as follows:

Definition 49. EAREP Variations

Classification of EAREP completely follows the EAREP defined as in Def. 48 only except condition (iii) which is substituted by the following accordingly:

1. EAREP-0: AUTH$\subset \{(u,v)|u,v \in U, checkAccess_{ABAC}(u,v) = true\}$
2. EAREP-1: $\{(u,v)|u,v \in U, checkAccess_{ABAC}(u,v) = true\} \subset AUTH$
3. EAREP-2: AUTH$\cap\{(u,v)|u,v \in U, checkAccess_{ABAC}(u,v) = true\} \neq \emptyset$ (assuming it is not EAREP 0 or1)

This simple variations make subtle differences in the solving technique of EAREP. The subsequent discussion will carry out solutions to all variations of EAREP 0-2.

Theorem 16. *EAREP-1 can be solved by using Algorithm 4.1.*

Proof

Trivial. By definition of EAREP-1, ABAC rule is required to be found for only the authorization set, $AUTH \setminus \{(u,o)|u,o \in U \times O, checkAccess_{ABAC}(u,o) = true\}$ as the rest of the authorizations are already covered by given $Rule'_{op}$. The parameters to be sent to Algorithm 3.1 are fairly straightforward, and the resulting rule is OR'ed with the given $Rule'_{op}$. Even in the the case of infeasibility, an exact infeasibility solution is provided in Chapter 3. Therefore, exact solution is guaranteed.

6.3.2 EAREP-0 Algorithm

In this subsection, the solution algorithm 6.2 for EAREP-0 is presented along with associated proofs and complexity analysis. This algorithm iterates through each AtomicExp in the given

Algorithm 6.2 EAREP-0 Solution Algorithm

Input: An EAREP-0 problem instance.

Output: Feasible/infeasible status and $Rule_{op}$.

1: $Rule_{op} :=$ NULL

2: **for** each Atomicexpr ar in $Rule'_{op}$ **do**

3: $SAT_{ar} := \{u1 \in U | ua \in UA \wedge \forall (ua, v1) \in uoLabel(ar).ua(u1) = v\} \times \{o1 \in O | oa \in OA \wedge \forall (oa, v2) \in uoLabel(ar).oa(o1) = v2\}$

4: $unAuth_{ar}$:= $SAT_{ar} \setminus AUTH$

5: $Auth_{ar}$:= $SAT_{ar} \setminus unAuth_{ar}$

6: **if** $unAuth_{ar}$ is $\emptyset$ **then** $Rule_{op} := Rule_{op}. \vee ar$; Continue

7: $UA' := UA \setminus \{ua | ua \in UA \wedge (ua, v1) \in uoLabel(ar)\}$, $OA' := OA \setminus \{oa | oa \in OA \wedge (oa, v2) \in uoLabel(ar)\}$

8: **if** $UA' == \emptyset \wedge OA' == \emptyset$ **then** Continue

9: ATTValSet := $\{(ua, ua(u1)) | ua \in UA'\} \cup \{(oa, oa(o1)) | oa \in OA'\}$ where $(u1, o1) \in Auth_{ar}$

10: Find $UL \subseteq ATTValSet$ such that $\nexists (u2, o2) \in unAuth_{ar}$ so that $PL \subseteq AttV(u2, o2)$ where for any $(u, o) \in U \times O$, AttV(u,o) = $\{(ua, ua(u)) | ua \in UA\} \cup \{(oa, oa(o)) | oa \in OA\}$

11: **if** UL is $\emptyset$ **then** Continue

12: **if** $Rule_{op}$ is Null **then** $Rule_{op} := ar \wedge \bigwedge_{(ua,val1) \in UL} ua(u) = val1 \wedge \bigwedge_{(oa,val2) \in UL} oa(o) = val2$

else $Rule_{op} := Rule_{op}. \vee ar \wedge \bigwedge_{(ua,val1) \in UL} ua(u) = val1 \wedge \bigwedge_{(oa,val2) \in UL} oa(o) = val2$

13: $Rule'_{op}$ substitutes $Rule_{op}$

14: $AUTH_{ABAC} = \{(u, o) | u, o \in U \times O \wedge checkAccess_{ABAC}(u, o) = True\}$.

15: **if** $AUTH_{ABAC} == AUTH$ **then return** "Feasible" and $Rule_{op}$ **else** Call Algorithm 3.1 for feasibility "status" for the $AUTH \setminus AUTH_{ABAC}$ and generate $Rule''_{op}$

16: **if** Status == "feasible" **then return** "Feasible" and $Rule_{op}.\vee := Rule''_{op}$ **else**

17: **return** "Infeasible" and $Rule_{op}.\vee := Rule''_{op}$

$Rule'_{op}$, and looks for a suitable $Rule_{op}$ which substitutes $Rule'_{op}$ in the given ABAC system to make it equivalent to the given EAS, and ensures the maximum utilization of $Rule'_{op}$. If no such $Rule_{op}$ is found, it looks into the Algorithm 3.1 to generate rule for the authorizations left out. If the Algorithm 3.1 returns infeasible as well, Algorithm 6.2 returns infeasible status.

As shown in Chapter 3, finding feasibility of ABAC RuleSet Existence Problem takes $O(|U| \times |O|)$ when OP is a singleton. Therefore, the dominant factor in EAREP-0 Algorithm 6.2 are the number of AtomicExpr in $Rule_{op}$ multiplied by complexity of line 10 where worst case could be exponential. However, that does not make the proposed solution worse than starting from scratch approach as in Chapter 3. There are multiple factors, i) there might be cases where Algorithm 3.1 is not needed at all, and ii) if line 6 succeeds more frequently, most of the complexities can be avoided. Therefore, a case-based comparative study is needed to analyze the performance.

Theorem 17. *Algorithm 6.2 is correct.*

Proof:

The proof is very similar to Theorem 13. Therefore, omitted.

Theorem 18. *EAREP-2 can be resolved by slightly modifying and combining the procedure of Algorithm 3.1 and 6.2.*

Proof

By investigation, trivial.

6.3.3 EAREP-0 Infeasibility Solution

This section discusses about an exact and an approximate infeasibility solution of the EAREP-0 problem.

6.3.4 Exact Solution

If Algorithm 3.1 fails, the exact solution is similar to the infeasibility solution defined in [13], adding an user attribute and an object attribute to the UA and OA, respectively. The complete

details to resolve infeasibility can be found in [13], hence, omitted.

Theorem 19. *Exact solution to ERREP-0 infeasibility always maximizes the rule quality metric.*

Proof:

By observation of the infeasibility solution procedure in 3, the additional attributes preserves the original Atomicexpr while additional (attribute,value) expression is AND'ed to it. Thereby, trivial.

6.3.5 Approximate Solution

Exact solution could be the most desired one, however, approximate solution proposed here will provide flexibility. The proposed solution in subsection 6.3.4 alters the given UA and OA which might not be desirable in some cases. Therefore, an alternate approximate solution has been proposed here which utilizes the aim of rule quality metric.

The proposed approximate solution is straightforward and does not add complexity at all. By the definition provided in chapter 3, conflict-free partitions are used to generate conjunctive terms which are 'OR'ed to the $Rule_{op}$. To keep it simple enough, we propose to discard the conflicted partitions completely and use the conjunctive terms generated from conflict-free partitions only. By observation, it can be clearly said, the proposed approximate solution will never allow more authorizations than the given AUTH, therefore, with the cost of restricting the power of $Rule_{op}$, the system is protected from unauthorized access. By investigation of the conflict-free partition and $Rule_{op}$ generation concept in Chapter 3, the approximate solution claim holds and the proof is trivial.

6.4 Case Studies

This section provides two detailed case studies for ERREP and EAREP, respectively.

6.4.1 ERREP Case Study

The following example shows a relevant case of the infeasibility solutions proposed for ERREP.

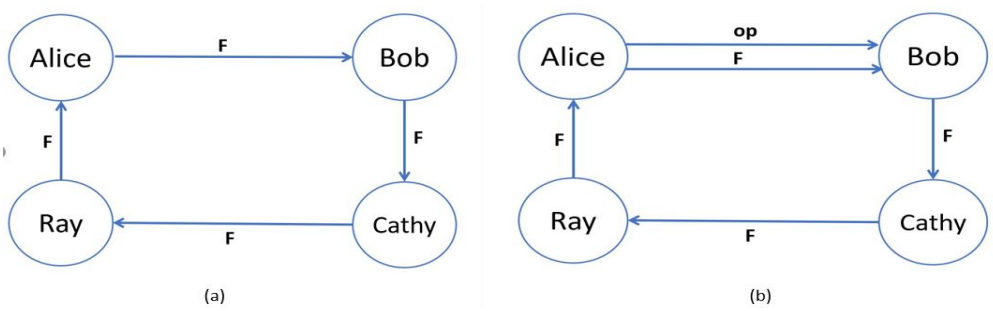

Figure 6.2: Infeasbility examples

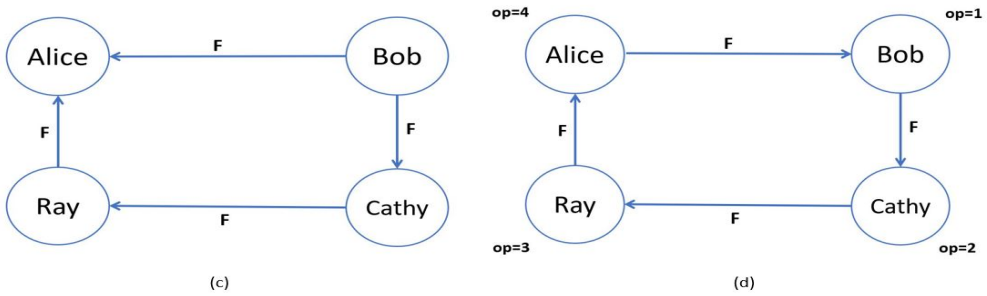

Figure 6.3: Infeasbility examples

First, consider the directed RG in Fig. 6.2(a) where V = {Alice, Bob, Cathy, Ray}, E = {(Alice, Bob, op), (Bob, Cathy, op), (Cathy, Ray, op), (Ray, Alice, op)}, and $\Sigma = \{F\}$. Let AUTH = $\{(Alice, Bob)\}$ and $Rule'_{op} = F$. Algorithm 6.1 fails here. Therefore, the exact solution is adding an relationship edge from Alice to Bob labeled as op as shown in Fig. 6.2(b). According to the aforementioned steps, $Rule_{op} = op$.

Second, consider the same Directed RG in Fig. 6.2(a) w.r.t. AUTH = $\{(Alice, Bob)\}$ and $Rule'_{op} = F$. As described by Def. 46, AuthPair = $\{(Alice, Bob)\}$ and unAuthPair = $\{(Ray, Alice), (Bob, Cathy), (Cathy, Ray)\}$ and the resulting G' is shown in Fig. 6.3(c). The topological sort algorithm returns a order Alice-> Ray-> Cathy -> Bob. According to that, the assigned priority values are shown in Fig. 6.3(d) and the final $Rule_{op}$ is (F, >).

6.4.2 EAREP Case Study

Here, an example of exact solution of EAREP will be shown first. Later, the proposed rigorous approximate solution will be shown on the same example. We reuse an example data from [13] for

Table 6.1: Example ABAC data

(a) UAValue	
User	uat1
u1	F
u2	F
u3	F
u4	G
u5	G

(b) OAValue	
Object	oat1
o1	F
o2	F
o3	F
o4	G

(c) Range of attributes	
uat1	$\{F, G\}$
oat1	$\{F, G\}$

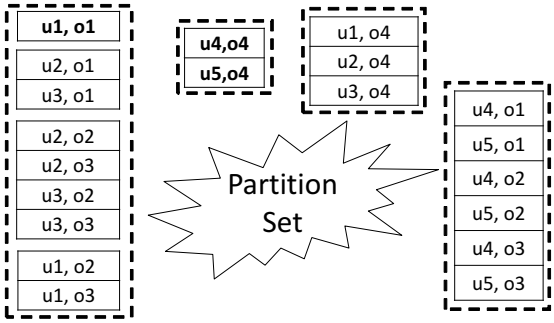

Figure 6.4: Refined partition set example.

this one.

Fig. 6.4 shows the resulting partitions for Table 6.1 where, bold black user-object pairs belong to AUTH with respect to OP = $\{op\}$ and rest are not. Here, U = $\{u1, u2, u3, u4, u5\}$, O = $\{o1, o2, o3, o4\}$, UA = $\{uat1\}$, OA = $\{oat1\}$, and Table 6.1 shows user attribute value assignment (UAValue), object attribute value assignment (OAValue), and range of attributes in (a), (b), and (c), respectively. The given ABAC rule, $Rule'_{op}$ is $(uat1(u) = F \wedge oat1(o) = F) \vee (uat1(u) = G \wedge oat1(o) = G)$. To make visual comparison, the dotted rectangles in Fig. 6.4 shows partition set P for Table 6.1 as defined in Section 3.4. Clearly, this is an EAREP-0 instance. As per Algorithm 6.2, $(uat1(u) = G \wedge oat1(o) = G)$ satisfies only authorizations belongs to AUTH, therefore, utilized in the rule. The other one, $(uat1(u) = F \wedge oat1(o) = F)$ satisfies both authorized and unauthorized pair, but no additional user and object attributes remain to be considered, therefore, ignored. The Algorithm 6.2 returns infeasibility as well, therefore, the rule quality measure is 0.5 here. In order to provide an infeasibility solution, the leftmost conflicted parition is refined into four sub-partitions using the exact infeasibility solution procedure.

For instance, given a conflict partition, P_i in Fig. 6.4 where only (u1, o1) belongs to AUTH with respect to op, it is refined into four new partitions. Initially, $uList_i$ is $\{u1, u2, u3\}$ and $oList_i$ is $\{o1, o2, o3\}$. According to Algorithm 3.2, $uList_i$ is further partitioned into $\{\{u1\}, \{u2, u3\}\}$. Similarly, $oList_i$ is further partitioned into $\{\{o1\}, \{o2, o3\}\}$. The resulting refined partitions has same PV, given by $\{(uat1, F), (oat1, F)\}$. According to Definition 22 and Algorithm 3.2, let exU value for $\{u1\}$ and $\{u2, u3\}$ be 1 and 2, respectively. Similarly, $\{o1\}$ and $\{o2, o3\}$ are assigned 3 and 4 for exO, respectively. Thereby, resulting unique PV_{new} value for the refined partitions are $\{$(uat1, F), (oat1, F), (exU, 1), (exO, 3)$\}$, $\{$(uat1, F), (oat1, F), (exU, 1), (exO, 4)$\}$, $\{$(uat1, F), (oat1, F), (exU, 2), (exO, 3)$\}$, and $\{$(uat1, F), (oat1, F), (exU, 2), (exO, 4)$\}$, respectively.

Based on this, two partitions $\{$(u1, o1)$\}$, and $\{$(u4, o4), (u5, o4)$\}$ are included in $Rule_{op}$ and rule quality metric is 1 now. Hence, $Rule_{op}$ is $(uat1(u) = F \wedge oat1(o) = F \wedge exU(u) = 1 \wedge exO(o) = 3) \vee (uat1(u) = G \wedge oat1(o) = G)$ and the RuleSet is $\{Rule_{op}\}$. In this example, both exU and exO are used for RuleSet Infeasibility Correction.

The approximate solution to this problem is straightforward, discarding the conflicted partitions generates the $Rule_{op}$ as $(uat1(u) = G \wedge oat1(o) = G)$.

6.5 Summary

This concludes the chapter. The extension works on the feasibility of ReBAC and ABAC policy mining are introduced and more analysis will be added in the future.

CHAPTER 7: CONCLUSION AND FUTURE WORK

This chapter summarizes the contributions made in this dissertation work. Furthermore, it discusses the possible enhancement and future directions briefly.

7.1 Summary

To the best of our knowledge, feasibility analysis in access control policy mining is the first such study made on the domain of access control policy mining. In order to analyze the feasibility, the first step is to identify the source and target access control system. After that, prime concern is whether supplementary information required, and a precise set of criteria or objectives. For example, given an RBAC System as source and ABAC system as target, supplementary data is optional [14], and the significant criteria is the target system is equivalent or approximate as compared to the source where entity id based rules are not allowed. This work presents an insightful study on the feasibility of ABAC, ReBAC, and AReBAC policy mining. Finally, an extension of the ABAC and ReBAC policy mining is proposed.

First work is on ABAC policy mining [13,14]: as shown in Chapter 3, ABAC RuleSet Existence problem is defined and analyzed here for EAS and RBAC as source access control systems. In order to devise a feasibility solution, set partition based approaches are utilized, and an idea of conflict-free partition is developed. Finally, ABAC rule is generated, regardless of infeasibility issues as we provide an additional attribute based remedy to the infeasibility problem arise here. Furthermore, associated proofs and complexity analysis are presented.

Second work is on ReBAC policy mining [11]: ReBAC policy mining is an emerging research field. As shown in Chapter 4, The ReBAC policy mining feasibility problem entails determining whether a ReBAC policy exists with the given structure and assumptions and, if so, how to obtain it. This topic is investigated in the context of various ReBAC policy languages, which differ in the types of relationships, inverse relationships, and non-relationships that can be utilized to construct the policy. We devise a feasibility detection technique and assess its difficulty. We show that when

we add more capabilities to our policy languages, they become more expressive. Various solutions are discussed in the event of infeasibility.

Third work is on AReBAC policy mining [12]: as shown in Chapter 5, this research investigates if an Enumerated Authorization System (EAS) can be converted to an AReBAC system using supporting attribute and relationship data. In this study [12], the Attribute-aware ReBAC Ruleset Existence Problem (ARREP) is explicitly defined for the first time, and it is solved algorithmically via complexity analysis. In the event of infeasibility, equivalent and approximate solutions are devised. Future enhancement directions are included in this study.

As shown in Chapter 6, fourth work is on a novel direction on the feasibility of ABAC and ReBAC policy mining: this work studies feasibility analysis of ReBAC policy mining from a novel direction, formally named as Extended ReBAC RuleSet Existence Problem (ERREP). Various directions of ERREP are discussed, defined, and solved algorithmically, along with complexity analysis. A similar study has been conducted on feasibility analysis of ABAC policy mining as well, Extended ABAC RuleSet Existence Problem (EAREP) and variations are defined and solved precisely. In both ReBAC and ABAC cases, notions of equivalent and approximate infeasibility solutions are developed. Finally, an elaborated case study has been included to clarify the concept.

7.2 Future Work

The study of access control policies is a relatively new area. As a result, the feasibility analysis of access control policy mining offers a novel route that includes certain interesting research directions as well as open research challenges that can be investigated. The research findings are attractive, and adopting them with real-world complex data is a major goal for the future. The final goal of this dissertation is to address the aforementioned concerns and to expand on them in numerous dimensions in order to enhance the newly developed feasibility problem domain.

- The future directions of Chapter 3 are as noted in [13, 14]. First things to be considered here is that whether we can improve the complexity of feasibility detection and infeasibility solution approaches. As ABAC RuleSet Existence Problem is defined now, variety of exact

and approximate solutions can be proposed, and real industry data can be used to validate and analyze the execution time as well. For example, as discussed in Chapter 3, a conflicted partition can be split into two or more fragments, however, a solution where we can always divide the conflicted partition into two conflict-free partition would be much better solution. Another aspect is whether addition of more attributes can impact the infeasibility solution approach. The impact of combining positive and negative ABAC rules could improve or degrade the current procedure as well. A statistical study on the feasibility of ABAC policy mining is the ultimate goal now to ensure the optimal outcome.

- As included in [11], the future direction of RREP will be noted here. In this study, the ReBAC RuleSet Existence Problem is being defined for the first time. Significant directions for improvement remain as open research problem. For example, ReBAC rule minimization, reducing exponential asymptotic complexity of the proposed algorithm, etc. Furthermore, adding more feature to the given rule structure, or the impact of proposing a new rule structure and a study on how the proposed algorithm is modified with it could be appealing reasearch problem. A more optimized question could be whether the feasibility detection process can be totally automated given any ReBAC rule structure. A study on RG with cycles and loops along with considering unlimited path length will be an interesting direction to consider. If significantly adjusted, our current technique would work with paths that include cycles and have a length restriction. We haven't looked into cycles that aren't limited in length. Extending the feasibility problem formulation as well as infeasibility solutions beyond user to user context is another key direction.

- As shown in Chapter 5 and [12], the ARREP has been introduced for the first time, to the best of our knowledge. A few directions for future enhancement will be addressed briefly. The proposed feasibility detection algorithm produces overall exponential asymptotic complexity. A significant form of improvement would be reducing the computation complexity. Extending the ARREP to other entities and consideration of various rule structures can be interesting research problems as well. More efficient infeasibility solution approaches (both

exact and approximate) can be proposed, which remained as open research problem. We consider cycle-free paths in ARG only. Including the cycle could be an interesting direction. Introducing different type of path in ARG, such as complementary path from [11], can optimize the current infeasibility solution approach. Additionally, the given ARG is static here. Accommodating the changes in ARG could be a valuable extension.

- As shown in Chapter 6, the proposed ERREP and EAREP offers some flexibility in redesigning the policy, especially when the existing policy is slightly modified. We leave the big case analysis with real world examples for future study.

This dissertation establishes the framework for feasibility in access control policy mining. It is now an open field where new issues can be incorporated as new access control models joins the domain.

BIBLIOGRAPHY

[1] Tahmina Ahmed, Farhan Patwa, and Ravi Sandhu. Object-to-object relationship-based access control: Model and multi-cloud demonstration (invited paper). In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pages 297–304, 2016.

[2] Tahmina Ahmed, Ravi Sandhu, and Jaehong Park. Classifying and comparing attribute-based and relationship-based access control. In *Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy*, CODASPY '17, pages 59–70, New York, NY, USA, 2017. Association for Computing Machinery.

[3] Glenn Bruns, Philip W.L. Fong, Ida Siahaan, and Michael Huth. Relationship-based access control: Its expression and enforcement through hybrid logic. CODASPY '12, pages 117–124, New York, NY, USA, 2012. Association for Computing Machinery.

[4] T. Bui, S. Stoller, and J. Li. Mining relationship-based access control policies. In *SACMAT*, pages 239–246, 2017.

[5] Thang Bui and Scott D. Stoller. A decision tree learning approach for mining relationship-based access control policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, SACMAT '20, pages 167–178, New York, NY, USA, 2020. Association for Computing Machinery.

[6] Thang Bui and Scott D. Stoller. Learning attribute-based and relationship-based access control policies with unknown values. In *Information Systems Security*, pages 23–44, Cham, 2020. Springer International Publishing.

[7] Thang Bui, Scott D. Stoller, and Hieu Le. Efficient and extensible policy mining for relationship-based access control. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, SACMAT '19, pages 161–172, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Thang Bui, Scott D. Stoller, and Jiajie Li. Mining relationship-based access control policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, SACMAT '17 Abstracts, pages 239–246, New York, NY, USA, 2017. Association for Computing Machinery.

[9] Thang Bui, Scott D. Stoller, and Jiajie Li. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers and Security*, 80:317 – 333, 2019.

[10] Thang Bui, Scott D. Stoller, and Jiajie Li. Mining relationship-based access control policies from incomplete and noisy data. In Nur Zincir-Heywood, Guillaume Bonfante, Mourad Debbabi, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 267–284, Cham, 2019. Springer International Publishing.

[11] Shuvra Chakraborty and Ravi Sandhu. Formal analysis of rebac policy mining feasibility. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, CODASPY '21, pages 197–207, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Shuvra Chakraborty and Ravi Sandhu. On feasibility of attribute-aware relationship-based access control policy mining. In Ken Barker and Kambiz Ghazinour, editors, *Data and Applications Security and Privacy XXXV*, pages 393–405, Cham, 2021. Springer International Publishing.

[13] Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. On the feasibility of attribute-based access control policy mining. In *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 245–252, 2019.

[14] Shuvra Chakraborty, Ravi Sandhu, and Ram Krishnan. On the feasibility of rbac to abac policy mining: A formal analysis. In *Secure Knowledge Management In Artificial Intelligence Era*, pages 147–163, Singapore, 2020. Springer Singapore.

[15] Y. Cheng, J. Park, and R. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pages 646–655, 2012.

[16] Y. Cheng, J. Park, and R. Sandhu. An access control model for online social networks using user-to-user relationships. *IEEE Transactions on Dependable and Secure Computing*, 13(04):424–436, jul 2016.

[17] Yuan Cheng, Jaehong Park, and Ravi Sandhu. A user-to-user relationship-based access control model for online social networks. In *Data and Applications Security and Privacy XXVI"*, pages 8–24, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[18] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Attribute-aware relationship-based access control for online social networks. In *Data and Applications Security and Privacy XXVIII*, pages 292–306, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[19] C. Cotrini, T. Weghorn, and D. Basin. Mining ABAC rules from sparse logs. In *EuroSP*, pages 31–46. IEEE, 2018.

[20] Aaron Elliott and Scott Knight. Role explosion: Acknowledging the problem. In *Software Engineering Research and Practice*, 2010.

[21] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC*, 4(3):224–274, 2001.

[22] Philip W.L. Fong. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, pages 191–202, New York, NY, USA, 2011. Association for Computing Machinery.

[23] Philip W.L. Fong and Ida Siahaan. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, SACMAT '11, pages 51–60, New York, NY, USA, 2011. Association for Computing Machinery.

[24] V. C Hu, D R. Kuhn, and D. F Ferraiolo. Attribute-Based Access Control. *IEEE Computer*, (2):85–88, 2015.

[25] Vincent C Hu et al. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800:162, 2014.

[26] P. Iyer and A. Masoumzadeh. Mining positive and negative attribute-based access control policy rules. In *SACMAT*, pages 161–172, 2018.

[27] Padmavathi Iyer and Amirreza Masoumzadeh. Generalized mining of relationship-based access control policies in evolving systems. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, SACMAT '19, pages 135–140, New York, NY, USA, 2019. Association for Computing Machinery.

[28] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.

[29] L. Karimi and J. Joshi. An unsupervised learning based approach for mining Attribute Based Access Control policies. In *IEEE Int. Conf. on Big Data*, pages 1427–1436, 2018.

[30] Leanid Krautsevich, Aliaksandr Lazouski, Fabio Martinelli, and Artsiom Yautsiukhin. Towards policy engineering for attribute-based access control. In *Trusted Systems*, pages 85–102, Cham, 2013. Springer International Publishing.

[31] Amirreza Masoumzadeh. Security analysis of relationship-based access control policies. CODASPY '18, pages 186–195, New York, NY, USA, 2018. Association for Computing Machinery.

[32] E. Medvet et al. Evolutionary inference of attribute-based access control policies. In *Evolutionary Multi-Criterion Optimization*, pages 351–365. Springer, 2015.

[33] B. Mitra et al. A survey of role mining. *ACM CSurv. 2016*, 48(4), 2016.

[34] D. Mocanu, F. Turkmen, and A. Liotta. Towards ABAC policy mining from logs with deep learning. In *IS 2015*, 2015.

[35] I. Molloy et al. Evaluating role mining algorithms. In *SACMAT*, pages 95–104, 2009.

[36] I. Molloy et al. Mining roles with multiple objectives. *ACM Trans. Inf. Syst. Secur.*, 13(4):36:1–36:35, 2010.

[37] Syed Zain Raza Rizvi and Philip W. L. Fong. Efficient authorization of graph-database queries in an attribute-supporting rebac model. *ACM Trans. Priv. Secur.*, 23(4), July 2020.

[38] R. S Sandhu et al. Role-Based Access Control models. *IEEE Computer*, (2):38–47, 1996.

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

[40] Ravi S Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[41] D. Servos and S. L. Osborn. Current research and open problems in Attribute-Based Access Control. *ACM Comput. Surv.*, 49(4):65:1–65:45, 2017.

[42] T. Talukdar et al. Efficient bottom-up mining of attribute based access control policies. In *IEEE CIC 2017, 339-348*, pages 339–348. IEEE, 2017.

[43] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: A formal perspective. *ACM Trans. Inf. Syst. Secur.*, 13(3):27:1–27:31, 2010.

[44] T. R. Weil and E. Coyne. ABAC and RBAC: Scalable, flexible, and auditable access management. *IT Professional*, 15(03):14–16, 2013.

[45] Z. Xu and S. Stoller. Mining attribute-based access control policies from RBAC policies. In *CEWIT*, pages 1–6. IEEE, 2013.

[46] Z. Xu and S. Stoller. Mining attribute-based access control policies from logs. In *DBSec*, pages 276–291. Springer, 2014.

[47] Z. Xu and S. Stoller. Mining attribute-based access control policies. *IEEE TDSC*, 12(5):533–545, 2015.

[48] Z. Xu and S. D. Stoller. Algorithms for mining meaningful roles. In *Proc. 17th ACM SACMAT*, pages 57–66, 2012.

VITA

Shuvra Chakraborty (shuvra.chakraborty@utsa.edu) is a Ph.D. student in the Department of Computer Science at the University of Texas at San Antonio. Her doctoral research works focus on the investigation of novel access control methodologies, development and deployment, especially, access control policy mining. She started PhD under the supervision of Prof. Dr. Ravi Sandhu since Fall 2016. She also holds masters and bachelors degree in Computer Science and Engineering from the University of Dhaka, Bangladesh. Before joining the UTSA, Shuvra has spent more than five years in teaching and research in several universities as a faculty, predominantly at the University of Dhaka, Bangladesh. During her PhD study, Shuvra has published and presented research works in recognized conferences. Apart from this, she has participated in reputable conferences such as BlackHat, WiCyS, GHC and so on as a student scholar. She is an active member of ACM, Alpha Chi, and Bangladesh Student Association at UTSA. Apart from this, Shuvra is very enthusiastic about the real-world cybersecurity problem, coding and traveling!