
بسم الله الرحمن الرحيم

ACCESS CONTROL POLICY VALIDATION METHOD

By (Muhammad Aqib)

A thesis submitted for the requirements of the degree

of Master of Science in Computer Science

Supervised By

Dr. Riaz Ahmed Shaikh

COMPUTER SCIENCE DEPARTMENT
FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY
JEDDAH – SAUDI ARABIA
Jumada Al-Awwal 1435H – March 2014G

ACCESS CONTROL POLICY VALIDATION METHOD

By

Muhammad Aqib

This thesis has been approved and accepted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

EXAMINATION COMMITTEE

	Name	Rank	Field	Signature
Internal Examiner				
External Examiner				
Advisor				

KING ABDULAZIZ UNIVERSITY

Jumada Al-Awwal 1435H – March 2014G

Dedication

To my beloved parents and teachers, who taught me to be ambitious…

To all who supported me to complete this work….

ACKNOWLEDGEMENT

First of all, I am thankful to Allah for giving me the opportunity to study for my master degree, for giving me the strength to complete this thesis, and for his endless blessing that kept me feeling all the time that he is organizing everything for me.

I would like to express my deepest sense of gratitude to my supervisor Dr. Riaz Ahmed Shaikh for his patient guidance, encouragement and excellent advice throughout this study. I appreciate his assistance in writing this thesis. Without his help, this work would not be possible.

I admire the help of many people who offered me valuable support throughout my study. I would like to express my special thanks to Dr. Ahmed Saeed Alzaharani, the head of the Computer Science Department, for his encouragement and support.

I express my deepest thanks to my colleagues who offered me valuable comments and provided me with the essence of their experience during this study.

I would also like to thank my family for the support they provided me through my entire life and in particular, I must acknowledge my parents and other family members, without whose love, encouragement, support and assistance, I would not have finished this work.

Access Control Policy Validation Method

Muhammad Aqib

Abstract

Inconsistency in access control policies exists when two or more than two rules defined in the policy set lead to the contradictory decisions. It makes it difficult for the system to decide which rule is applicable to the current scenario and hence make the system vulnerable to the unauthorized use. Different inconsistency detection methods have been proposed by researchers. However, those suffer from various limitations, such as, inefficient handling of continuous attribute values, unable to handle Boolean expressions and the ignorance of contextual attribute values such as date, time etc. In order to overcome these limitations, a new algorithm is proposed that detects the inconsistencies in the policies using multi-terminal decision trees.

A comprehensive state-of-the-art survey is also provided in this work on the existing access control policy validation techniques, which is currently not available in the literature. This survey includes taxonomy and qualitative comparison. Taxonomy is used to identify the current trends and qualitative comparison is used to identify the pros and cons of the existing schemes in an extensive manner.

Based on the proposed algorithm, a software tool named "ACP Validation Suite" is developed. This tool first takes the access control policies defined in the XML. After that it displays them in multi-terminal decision tree form. On this decision tree, the tool will execute the proposed policy validation algorithm and display all inconsistent rules present in the policy set.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Background

1.1.1. An Overview of Access Control Policies

Access to resources in enterprise environments is restricted by applying different mechanism and every user is not allowed to access each and every resource or information present in those systems. For example, in a university, students can access the system to view their attendance, marks, grades and courses available for registration etc., but they are not allowed to mark their attendance, change their marks and to add more courses in the list available for registration. All this is done by applying a mechanism to control the access of users to the different resources of the system. For this purpose, different rules are defined by the system administrators to restrict the users' access to resources. These rules are defined under different kind of policies which are applied for this purpose and are known as the access control policies.

1.1.2. Types of Access Control Policies

There are many types of access control models but mainly they can be categorized into three main classes [1]: 1) Discretionary Access Control (DAC), 2) Mandatory Access Control (MAC) and 3) Role-Based Access Control (RBAC). In this section we will briefly describe these models. Other types of these policies include Attribute Based Control (ABAC) [62], Workflow Based Access Control [63] and Chinese Wall [64], but in the following paragraphs we will discuss the above three types in detail.

1.1.2.1 Discretionary Access Control (DAC)

In DAC, the access to any resource in the system is granted on the basis of the identity of the user. For example, the user is supposed to enter user name and password. It is known as discretionary because in this model a user may transfer his ownership to some other user. The access matrix model is a common example of the DAC which was first proposed by the Lampson [39] in which the authorizations holding by the user at different states are represented as a matrix. This idea was further refined by Graham and Denning [40] and later on by Harrison, Ruzzo and Ullmann [41].

1.1.2.2. Mandatory Access Control (MAC)

In MAC, certain rules are defined by the administrators of the system and access to different resources is granted on the basis of those rules. Multilevel security (MLS) policy is the most common form of MAC and it is based on the security clearance level

of subjects and objects in the system [1] [42]. Bell-Lapadula model [43] (for confidentiality) and Biba model [49] (for integrity) are the two common examples of MLS models.

1.1.2.3. Role-Based Access Control (RBAC)

RBAC is an alternative to both DAC and MAC and is commonly used to define the access control policies. It divides the privileges amongst different roles and every user is granted access to resources according to its role in the system. For example, student in a university can access his attendance record of a student but he cannot modify it. Similarly, he can see his grades list of different courses but cannot make any changes in it. Only teacher can enter the attendance of the students and can enter and update student's grade. So the access is granted to the users according to their responsibilities in the system [44] [45] [46].

1.2. Motivation and Problem Statement

Access control policies play an important role in security of enterprise applications. With the use of this concept, the system administrators can define the rules to grant access to users to access the specific resources if they fulfil the requirements defined in the policy rules. So the idea works fine if the rules are clear and error free. Problem arises when administrators are supposed to define rules for large number of users and when the access conditions may also change dynamically. There may be overlapping conditions in the defined rules and some of the defined rules may contradict with each other in result of those conditions. The existence of such contradictory rules is considered as *inconsistency problem*. In order to remove inconsistencies in defined rules, the rules should be validated before implementation.

For this purpose, researchers have proposed different validation mechanisms [2] [3] [6] [7] [8] [9] etc. but they suffer from various limitations. These limitations include inefficient handling of continuous attribute values, unable to handle Boolean expressions and the ignorance of contextual attribute values such as date, time etc. So there is a need of a validation method which could handle inconsistency problem by focusing all these limitations as well.

1.3. Research Objectives:

The objective of this research is to develop a new inconsistency detection method for access control policy sets that overcome the above-mentioned limitations of existing policy validation methods.

1.4. Contribution

In this thesis, first, we have provided comprehensive survey on access control policy validation techniques. To the best of our knowledge, this is the first comprehensive survey on policy validation techniques. This survey contains taxonomy for access control policy validation techniques, qualitative comparison of the existing policy validation techniques, and trend analysis, which identifies most common and new emerging techniques used for the policy validation.

After the comprehensive analysis of the existing schemes a new access control policy validation method is proposed. The proposed method not only detect inconsistencies in access control policies but it also provides resolution mechanism.

The proposed method contains the following unique features:

- It provides support for both continuous and discrete attribute values.
- It can detect inconsistencies in both static and dynamic policies.
- It can handle Boolean expressions defined in the rules.
- It can also handle contextual attributes like date, time etc.

In this thesis, theoretical and implementation evaluation is also performed for the proposed policy validation algorithm. Results show that the proposed algorithm is not only efficient but also it is easy to implement.

1.5. Thesis organisation

This thesis is organised as follows:

Chapter 1 introduces the research subject, providing a simple overview of the problem, and addressing the objectives and motivation of this research.

Chapter 2 provides the review of literature in seven main sections. Validation techniques taxonomy has been given in this section and all the techniques proposed by researchers have been categorized according to the approach used for validation. Trend analysis has also been given to present the trend of researchers in using different techniques for validation purpose.

Chapter 3 presents the proposed solution for the detection and resolution of inconsistency in access control policies. It also provides the definition of inconsistency and explains it with the help of detailed example.

Chapter 4 provides the implementation details of the proposed algorithm. The tool developed for this purpose has been discussed in detail. This include the architectural design details and all the relevant informations.

Chapter 5 presents the analysis of the proposed algorithm. This chapter has two main sections. In first section we have given the qualitative comparison of the proposed technique with the other techniques proposed by other researchers. In the other part of this chapter, the complexity analysis of the proposed algorithm has been discussed in detail.

Chapter 6 concludes the work that has been done in this study and proposes recommendations for the future.

Chapter 2

Review of Literature

Different approaches have been adopted by the researchers to address the ACPs verification and validation issues. In this section, we have presented some of the methods and frameworks proposed for policy validation. We have classified the proposed methods into the following six categories.

- Decision Diagram Techniques
- Mining Techniques
- Model Checking Techniques
- Formal Methods
- Matrix-based Approaches
- Mutation Testing Approaches
- Other Techniques

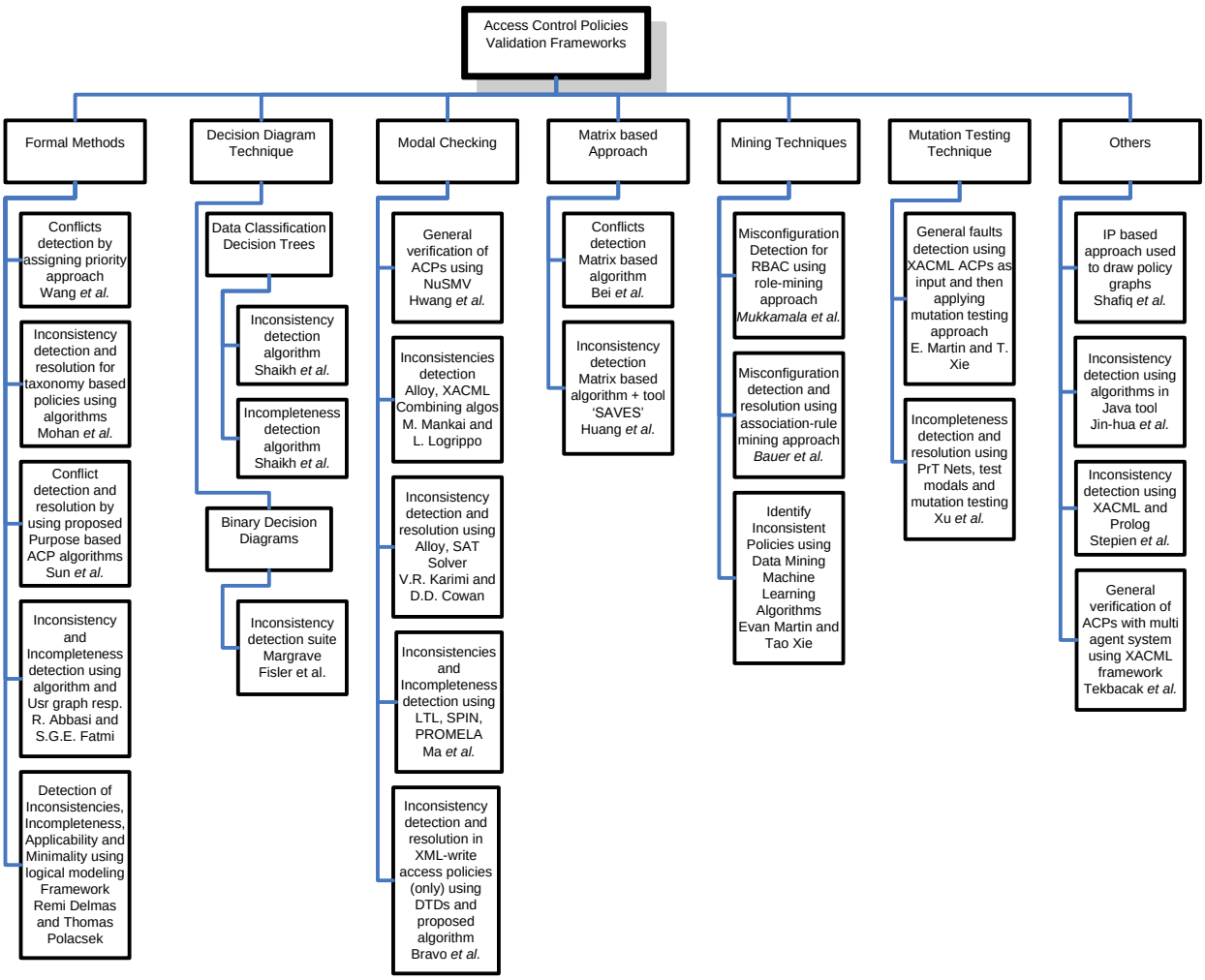


Figure 2.1. Classification of different approaches for validation of ACP

2.1. Decision Diagram Technique

Decision diagrams are the tree like structures having multiple decision and terminal nodes. These diagrams are useful to validate the access control policies because these can be used to separate the data on the basis of their attribute values. In this article we have divided the work done by different researchers in this area in the following two subcategories.

2.1.1. Data Classification Techniques

Some authors have used data classification techniques to identify the inconsistency and incompleteness problems in the ACPs. They have used different algorithms like ID3 [27], C4.5 [28], and ASSISTANT 86 [29].

Shaikh *et al.* in [3] have discussed the inconsistency issue in detail and have proposed an efficient mechanism to detect inconsistency in ACPs. In presence of different data mining techniques like ID3 [27], C4.5 [28] and ASSISTANT 86 [29], the authors have selected C4.5 data classification technique for this purpose and have made some modifications to make it more progressive and effective for consistency detection. According to authors the access control policies or rules are collection of attributes. Attributes are classified as non-category which is decision making attribute like subject, role, action etc. and category attributes which defines the class of rule which it belongs e.g. allowed, denied.

The authors have categorize the inconsistency into two types: a direct inconsistency which occurs when two or more rules present in the same policy set lead to contradictory conclusions and the indirect consistency where two or more rules belonging to different policy sets lead to contradictory conclusions.

There are two main steps of the inconsistency detection strategy adopted by the authors. In first step they need to create a complete decision tree. The C4.5 algorithm has been used to create decision trees, but the original algorithm creates optimize tree in which all the attributes may not present. For inconsistency detection, a decision tree must contain all attributes. So authors have made some changed in the C4.5 algorithm that ensures that all attributes must be present in the decision tree. After creation

of the decision tree, an algorithm is used to detect the inconsistencies. This algorithm first checks the terminal or leaf nodes of each branch. If any leaf node contains more than one category attributes, it means that inconsistency exists in rules represented by that branch. So all the attributes of that particular branch are fetched and by searching the attribute values in the policy set, all the rules in the policy set containing those attribute values are highlighted as inconsistent. If all the terminal nodes contain only one category attribute value, then the policy is considered to be consistent.

The authors have provided different examples of both direct and indirect inconsistencies which shows that the proposed solution can efficiently detect inconsistencies in both cases. Another important feature of the proposed system is its linear computational complexity whereas the many other methods, based on formal logic have exponential computational complexity.

Shaikh *et al.* in [4] have provided a mechanism to detect incompleteness in ACPs using data classification techniques. According to them, completeness checks are generally performed manually by the administrators and completeness is sometimes achieved by adding negative authorizations and sometimes access is denied in unspecified cases. Data classification algorithms used by authors for incompleteness detection are Limited Search Induction Algorithm (LSIA) [32], C4.5 [28] and ASSISTANT'86 [29] with some modifications.

The incompleteness detection mechanism proposed by the authors consists of five steps. Initially in the first step, rules in the ACPs are classified according to different resources. This separates the rules defined for different resources to avoid conflict in rules defined for different resources. Secondly, define non-category attributes for each resource. The values for different attributes which are present in the rules for different resources are fetched in the third step. In step four, different data

classification algorithms are used to create decision trees for each resource. In step five, Incompleteness algorithm is applied on the decision tree. This algorithm checks the terminal nodes of the decision tree. If the terminal node does not contain any category attribute value it means there is incompleteness in the policy set.

The authors have applied these techniques on two different scenarios and have concluded that all the data mining algorithms are not suitable for incompleteness detection in ACPs. They have proved that the modified (extended) form of C4.5 algorithms is best for this purpose. Furthermore the modified version has reduced ordered complexity as compared to the original algorithm.

The proposed method detects only the incompleteness in ACPs and it does not provide any solution to remove the incompleteness problem. It deals with discrete values and not useful in the case of continuous attribute values neither it deals with Boolean attribute values.

2.1.2. Binary Decision Diagram Technique

Different tools have been developed by researchers to validate access control policies using binary decision diagrams, like Fisler et al. [30] have developed a tool to analyze the role-based access control policies.

In [30], Fisler *et al.* have used multi terminal binary decision diagrams for the verification and validation of access control policies. They have presented a software Margrave, which can be used for the validation of the access control policies. A verifier has been used in Margrave to analyze the policies. This component takes access control policies written in XACML as input and generates different types of decision diagrams, which are further used in the verification process.

Margrave basically is divided into two components. It has a verifier, as discussed above and the other component is used for the change-impact analysis. It compares two policies changed due to some reasons and provides a summary also provides the facility to verify the changed properties of compared policies.

Margrave supports the XACML rule-combining algorithms which include: first-applicable, permit-override and deny-override. These are used to combine rules from different policies. According to the authors, Margrave can also use EPAL [47], which is another access-control language by IBM. It uses multi-terminal binary decision diagrams (MTBDD) to represent the access control policies and the outcomes of these policies (permit, deny, not-applicable) are represented by the terminal nodes. CUDD [48] has been used to implement MTBDDs.

To test the performance of this tool, the authors have evaluated the access control policies of a research paper submission website. They translated its policies in XACML and verified using Margrave. Both of its phases; policy querying and verification, and change-impact analysis were completed in very short time and it was scalable with respect to the memory usage as well. It also pointed out the lapse in security policies. But it has some limitations as well. It is useful to detect the inconsistencies in discrete and static data. It is not helpful in case of dynamic data neither it supports the contextual attributes. It also deals with the inconsistency problem only and the incompleteness problem has not been addressed in it.

2.2. Mining Techniques

Data mining techniques are the techniques used to extract different data patterns from a large amount of data and to convert them into the required format to

make them useful in different environments. In the context of access control policies validation mechanisms, these techniques have been used by different researchers and different tools have been developed using these techniques.

In [51], Mukkamala *et al.* have proposed a method to detect and resolve the misconfigurations in RBAC policies. They have used the following two terms to discuss the misconfigurations in the access control polices: under-privileges and over-privileges. Furthermore two approaches are discussed by the authors that are normally used to address these problems: top-down approach and bottom-up approach. The authors have used the bottom-up approach, also called role-mining problem.

Authors have used a tiling approach proposed in [52] to discover roles by using privileges. Their work is the solution to the role mining problem described in [52]. In this approach two algorithms are applied which use a matrix to represent users and privileges in rows and columns respectively. The intersection of rows and columns is represented by 1 if a user has a corresponding privilege and by 0 if it is not. Rectangular areas in that matrix with contiguous 1s are the tiles and represents different roles. Two algorithms are applied to get the minimum number of tiles (roles). According to the authors, there are four possible cases which arise from this situation and different solutions have been provided by the authors for those four cases to avoid misconfigurations in policies.

In that paper all the four possible cases to deal with under-privileges and over-privileges have been discussed and to test their results, forward-engineering and reverse-engineering approaches have been used. They are confident that their proposed role-mining approach can effectively use to deal with misconfiguration in RBAC policies. However, their approach has a very limited scope and it only deals with

simple policies without the involvement of conditions or contextual attributes like time, date etc.

Bauer *et al.* in [53] have proposed a method to handle the misconfigurations in access control policies. They have used the association rule mining approach and have provided the way to first detect and then to resolve those misconfigurations. Their approach mainly relies on the inference mechanism and uses if-then-else rules structure. Their approach tries to identify the misconfigurations in the policies in advance before they could create any trouble for the users and then tries to eliminate them in a second step.

In association rule mining technique, mainly attribute values which are normally set to true or false are used to identify the attributes which exists in multiple records. The attributes in this technique represent the resources and their values represent their existence or absence in a particular record. Subsets of these attributes are further used to construct the rules which describe that if first attribute (premises) of a record is present in a record, then the last attribute (conclusion) should also be present in that record. For the evaluation of this method they have used a system which was already implemented in their office for the last two years.

Apriori algorithm [54] has been implemented by the authors to apply association rule mining approach. If a user accesses some resources of a record, the attribute values to those records are set to true. The concept of premises and conclusion describes that if a user can access the premises of a record but the conclusion is not present then this is a misconfiguration. Furthermore a feedback system has also been developed which counts the number of correct or incorrect predictions. For every correct prediction, 1 is added to the count and it is decremented in case of a wrong prediction.

To evaluate the performance of the system, policies are divided into four categories: implemented policy, intended policy, exercised policy and unexercised policy and the performance of the system has been evaluated according to these four types. After the detection of misconfigurations, techniques to repair them have also been discussed in detail which states that any other authorized member may correct that, instead of only the administrators.

This technique is useful in detecting and resolving the inconsistencies in access control policies but its scope is very limited. It only takes the policies into account having multiple attributes with only Boolean values. Although it is dynamic in the sense that any user can delegate his rights to any other user but it depends on the inference mechanism. Contextual attributes like time, date etc. also seem beyond the scope of this approach.

Evan Martin and Tao Xie in [55] also have presented data mining approach for the verification of access control policies. They have tried to find out the differences between the policy specifications and their functionalities. For instance, they have given an example of the access control policies defined to grant access to the users in the university in such a manner that students should not be able to edit their grades. However, due to some specification problems students are allowed to edit the grades. Authors want to identify these problems using some requests which could expose those sort of bugs in the policies.

They have developed a tool which generates requests to be sent to the system. This tool supports two techniques: first one is to simply identify the XACML request documents and the other one constructs a request factory by inspection which then generates the requests on demand. Sun's XACML implementation [57] is used for the

evaluation of the generated requests. Weka[56] is used to apply machine learning algorithms for data mining tasks.

The solution proposed by the authors is applicable if all the attributes have limited values. For example, if a policy has three attributes like subject, object and action then the values of all these attributes should be finite. Furthermore, it generates possible combinations during request generation. Moreover, it is limited to the discrete values only and no contextual attributes are supposed to be included in the policies.

2.3. Modal Checking

In many approaches, the authors have used some modeling tools to validate the ACPs. These tools have their own validation criterion and use specific language like XACML [50], [60] for policy specifications. In this section we will discuss all those mechanisms which use modal checking tools.

In [5], Hwang *et al.* have addressed the important and challenging task of defining Access Control Policies to gain access to different resources in enterprise applications. Due to the existence of a large number of rules and complexity of the access policies, it is very important for the policy authors to conduct policy verification and validation to ensure the correctness of policies according to policy specifications. Access Control Policies Testing (ACPT) is a tool developed by the authors to address the problems of the policy authors. This tool helps the policy authors in policy modeling, implementation and verification. ACPT not only generates enforceable policies in XACML format using policy requirements but also performs the static and dynamic verification of these policies to reduce conflicts and faults in these policies.

There are four main components of this tool, named as policy modeling, static verification, dynamic verification and policy implementation. Policy modeling is the first component of this system not only helps the policy authors to create policies based on Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC) and Multi-Level Security, but also helps them to add, delete and modify the existing policies and their attributes. It generates a policy in the form of XACML and maps the input policy to the corresponding XACML attributes and includes conditions in the form of Boolean functions. It also performs static and dynamic verification on these policies. SMV specification language is used to represent the policies and their properties as a corresponding finite state machine (FSM). A symbolic model checker NuSMV [33] can check whether a policy is true or false. In this way it identifies the problems in the policies but does not provide any solution for them. It takes three attributes subject, action and object to perform combinatorial tests during dynamic testing which is a process to assure the correctness of a policy.

This tool helps the policy authors to specify policies based on different access control policy models using the graphical interfaces. It also allows them to modify or delete the existing policies using editors.

This tool is very helpful in generating policies based upon the policy requirements but its scope is very limited. It does not identify an inconsistency or incompleteness problems. Although it allows conditions (Boolean expressions) but its testing mechanism only verifies the simple policies which does not involve any contextual attributes like time, location.

M. Mankai and L. Logrippo in [7] have proposed a system to detect inconsistencies and conflicts in the access control policies. They have used a standard

logic model checking tool Alloy [34] [35] [36] for this purpose where the Access Control Policies (ACPs) have been written in XACML.

A logical model of XACML has been given in this paper which further has been translated into Alloy for inconsistency detection. In a brief discussion of logical model of XACML, policy modeling structure (functions and relations) and access control constraints have been discussed in detail. Modeling structure includes the definition and mapping of attributes, values, subjects, resources, actions, requests, targets, effects, combining algorithms, policies and policy sets. In PDP, every request to access some resources has to obey the constraints imposed by targets and conditions. Evaluation of targets against request and response of rules and policies include the target verification, rule response, policy response and policy set response.

In the proposed system the logical model is translated into the Alloy which is structural and declarative language. They have used the Alloy Analyzer [37] for the analysis and verification of Alloy model. The alloy structure uses the concept of signatures (a type in Alloy, same like a class in other languages) and relations (relates signatures and their instances). Functions are used for mapping of one signature to only one instance of the other signature. Every set in XACML is defined by a signature which is related by relations and functions. Signatures are declared to define the set of policies and the set of subject, object and action. These signatures contain different functions and facts to map different relations defined in the logical model. Predicates which are used to return true or false (if some target's subject, object and action matches or not to the corresponding values in a request) are also defined for the target verification purpose. If a target matches a request, the response defined in logical model is returned.

The authors of this paper have proposed a logical model which defines the policies in XACML and this logical model is translated into Alloy for inconsistency detection. This model has some limitations. It does not include any type of conditions and contextual variables. Further it deals with the static data and no dynamic change has been handled in this system. It has a high computational complexity and authors are not sure whether it will always complete in reasonable time or not.

V.R. Karimi and D. D. Cowan in [9] have specified ACPs related to Resource-Event-Agent (REA) business processes and the verification of these policies in conjunction with REA is the main purpose of this work. According to them, ACPs are not same for all the organizations and within the organization in different time slots. It is difficult to analyze all the policies because of their complexity.

The REA model contains two groups of business process, exchange and conversion. Sales and loans are the examples of these two exchange processes.
The alloy has been used for specification and verification of ACPs. Alloy Analyzer translates the rule into the Boolean formula and SAT solver produces the solution for this formula. SAT solution is further translated into the Alloy language by Alloy Analyzer [37]. The authors have created the directed graphs using the Alloy's meta-model option. They have examined an example which includes the ACPs in addition to a REA business process.

The proposed solution is suitable for the specific scenarios of same kind. Furthermore only one process has been used in this process. It seems to be a complex model because undesired results have been obtained by adding only one policy. It may work in small scope and with the increase in scope the chances to find errors decrease. Ma et al. in [10] have proposed a model checking based method for the validation and verification of security policies. For this purpose they have used linear temporal logic

(LTL) to describe the properties and the model checker SPIN has been used for the verification and validation of security policies.

In model checking, the properties are described using temporal logic formula and the system behaviour is represented as the transfer structure.

To represent the system behavior, the finite-state reachability graph is used which is described as Kripke structure. The LTL formula, used to describe the properties is converted to Buchi automaton. The system behaviour is represented by infinite strings of state labels and the LTL property automaton accepts only those state labels which are models of the formula.

The SPIN model checker has been used in this method which supports the design and validation of asynchronous systems. It accepts the design specifications written in PROMELA and LTL syntax is used for correctness claims. In model checking the result either validates the property or it returns a counterexample for any violation occurred.

In the proposed system the authors have indicated that for the security of the information systems, it is needed to validate the security requirements, security policies and the security solution for the requirements. Consistency and completeness should be ensured for all these areas. Security validation and verification includes the Kripke structure and LTL formula which are converted to Buchi automaton by SPIN model. Validate sequences are also generated for the security verification and validation purposes and a framework for this purpose has been presented by them.

Verification criteria have been set for the validity and reliability of the model checking to test the completeness and consistency problems. It has also been mentioned that in case the system does not match the property, a counter example is provided.

Bravo *et al.* in [16] have discussed a consistency detection and resolution method called ACCOn. According to them, they can use this method to detect inconsistencies in the XML write-access control policies defined using document type definition (DTD). Further, they have modified an existing algorithm to remove the inconsistencies form the policies.

As a DTD can be represented as a directed acyclic graph called a DTD graph. They have used this graph to represent different security policies and have defined some rules to represent the security policies using these graphs. In ACCOn model the authors have considered the delete, replace and insert update operations. To perform all these actions they have defined some rules which allow the user to update the tree as desired according to the access rights to perform an action. They have set different notations for different policies allowing an operation or disallowing it. If a policy defined over the DTD does not allow a forbidden update operation through a sequence of allowed operations then it is considered as consistent.

To test a given policy for insert or delete inconsistencies, a marked graph of XML DTD has been built. By applying the rules defined by the authors we can detect the insert/delete inconsistencies in that policy. To detect the inconsistencies in the replace operation another graph is used.

To resolve the inconsistencies they have proposed an algorithm that takes the replace graph as an input for a graph and runs a modified version of the Floyd-Warshall algorithm. The alternative algorithm for this purpose is named as Set Cover algorithm and uses Floyd-Warshall algorithm.

This paper focuses on detecting inconsistencies of specific type which are related to the XML Write-Access security policies. It is static and is applicable for discrete data only. No contextual attributes have been considered in this case.

2.4. Formal Methods

Methods for the validation of access control policies involving mathematical concepts and techniques are considered as formal methods. Some techniques include algorithms, based upon different types of mathematical concepts are usually considered as the formal methods for access control policy validation mechanisms. Many researchers have used different mathematical concepts in their proposed access control policy validation mechanism. Some of those techniques are discussed below.

In [2] Wang *et al.* have discussed the conflicts in ACPs which according to them occur when a set of policies is satisfied simultaneously and the system cannot take decision. The components of the information system described here are subjects, groups, objects, types, roles and actions. Every subject is related to a group, an object is related to a type. A group has some privileges and a subject belonging to this group can perform an action on an object or type of object using these privileges and the roles assigned to it. This model supports the triple tuple policy specification i.e. (subject, action, and object). Authors have categorised the conflicts into three types: modality conflicts, redundancy conflicts and potential conflicts. According to the authors, modality conflicts are the inconsistencies which may arise when two or more policies with opposite modalities refer to the same authentication subjects, authentication actions and authentication objects. Redundancy conflicts occur when we try to resolve modality conflicts and assign priorities to other policies in the set. In contrast to these two conflicts, potential conflicts occur when two policies have overlapping conditions. In this case two policies have no modality and redundancy conflicts, but when simultaneous satisfaction of their associated conditions cause modality or redundancy conflict. To resolve the modality conflicts, the conflicting policies are assigned

priorities so that the policy with the higher priority takes precedence. Global assignment of priorities to prioritized ACPs can also resolve the modality conflicts effectively. On the other hand, principle of specific take precedence is used to resolve redundancy conflicts. If a policy is a redundant policy, it is assigned a higher priority. For any two policies P_i and P_j, P_i should be assigned higher priority according to principle of specific take precedence. According to this work, priorities will be swapped between P_i and P_j and then check ACP_j, which points out any kind of redundancy and hence this way the redundancy conflicts can be removed. Potential conflicts are the conflict between the conditions of two policies, so system security officers (SSO) add permissions or prohibition to the associated conditions. Now according to the proposed method, if there is no potential conflict in PACPs, then the PACPs cannot derive any actual conflict. The author hopes that resolving these three types of conflicts by using the proposed solution ensures the error-prone implementation of ACPs.

Mohan *et al.* in [11] have discussed taxonomy-based ACPs for biomedical databases. In this paper the authors have discussed about the detection of inconsistencies in ACPs and information inference vulnerability detection and also have provided their solution. They have proposed dynamic conflict detection and resolution strategies for hierarchical data. In their work, an algorithm has been proposed to detect the inconsistencies in the taxonomy based data and another algorithm has been proposed to detect and resolve the inference attacks.

According to a tree structure, the authors have divided the nodes in that tree into class-subclass hierarchies. According to them e.g., suppose flu is a disease and all the types of "flu" are the subclasses of the class flu and are represented as the child

nodes in that tree. So the policy applied to a class or parent node will be applicable to the subclass or child nodes as well.

In taxonomy based authorization policies, the authors have addressed the conflicts among the different hierarchical levels in the resource tree and the detection of inconsistencies in authorization policies for inference related nodes. Their approach does not resolve these inconsistencies but provides a mechanism to detect them.

Two algorithms have been designed to detect inconsistencies and inference conflicts. Both these algorithms have been implemented using Java language and XACML has been used for policies. Furthermore, real data obtained from the NIH sponsored i2b2 project [22] has been used for evaluation. The performance of the system has been measured by measuring the time spent to run the algorithms for different sizes of the trees used as the input trees. It has observed that the total conflict handling time for a node is directly proportional to the number of nodes in the sub-tree.

The scope of this research is limited to the taxonomy based authorization policies only. It deals with the discrete data and the contextual attributes (e.g time) have not been considered in the proposed solution. It only detects inconsistencies but do not resolve them. The incompleteness problem is also not addressed.

Sun *et al.* in [13], think that access control is an important topic but the importance of privacy yet has not recognized in the traditional access models. In this paper they have tried to bridge the gap between the private information protecting technology and access control models. In this paper they have discussed the Usage Access Control (UAC) model which consists of eight components: subjects, subject attributes, objects, object attributes, rights, authorizations, obligations and conditions. As compared to UAC they have designed an extended PAC model to protect the

important information from unauthorized use. PAC is a purpose based access control technology for the challenges of privacy violations which is an important issue nowadays.

This paper focuses exclusively on how to specify and enforce policies for authorizing purpose-based access management using a rule-based language. For this purpose a framework has been proposed. This framework deals with purpose and data management purposes have been organized in a hierarchy and each data element is associated with a set of purposes.

For purpose based access control policy the authors have divided the purpose (a reason for data collection and data access) into two categories: Intended purpose which is related to data and regulate data accesses and Access purpose which is related to access the data. Intended purpose has further been divided into the Allowed Intended Purpose (AIP) and Prohibited Intended Purpose (PIP). In the proposed framework a policy (rule) is a tuple of the form (Subject, Action, Resources, Purpose, Condition, Obligation) where purposes are applied to achieve fine-grained policies. Purposes have been represented in a hierarchical structure and it is possible that conflicts may occur in the purposes of two different policies. To detect the conflicting purposes and conflicting policies, two algorithms also have been presented where first algorithm detects the conflicts in purposes of different policies and based on the first algorithm, the second algorithm detects the conflicts in the access control policies.

R. Abbasi and S. G. E. Fatmi in [15] have discussed different approaches followed by different authors in the field of information security by implementing different access control policies to restrict the users from unauthorized access of resources. In this paper, they have proposed a solution to detect the inconsistencies, incompleteness and preservation of safety and aliveness problems in the access control

policies by using the reasoning method which is used in software engineering. They have defined a security policy by using formal specifications and has validated this policy by using the executable specification method.

The concept of executable security policy (ESP) has been introduced by the authors for the validation of security policies. It uses a specification language and this proposed model uses PROMELA as a source of inspiration. The proposed validation process consists of three steps which are: (1) consistency proof, (2) completeness proof and (3) the SP properties preservation.

The authors have described some concepts regarding the consistency security policies and have provided an algorithm which uses those concepts and tests the security policies for inconsistencies. To test the SP for the completeness, the reachability analysis of the state model has been used and two reachability graphs have been used for this purpose. Furthermore, lifeness property and safety property have been discussed in detail. The concepts of exhaustive set, uniformity hypothesis and regularity hypothesis have been introduced to derive a finite SP reachability graph.

This paper deals with the security policies related to the firewall only. It has used the reachability graph for this purpose and security model is inspired by PROMELA. This model can be used for the detection of inconsistency, incompleteness and SP preservation verification.

Rémi Delmas and Thomas Polacsek [58] have proposed a logical modelling framework to find the inconsistencies and incompleteness in the access control policies. Providing a mechanism for the detection of these two properties, they have introduced two new properties, applicability and minimality and their proposed technique is capable to detect these two properties as well.

In the proposed framework, authors have used the MSFOL (many-sorted first order logic) [59] logical framework for this purpose. They have derived another logical framework from the MSFOL named PEPS (Peps for Exchange Policy Specification). So according to them, the PEPS signature is basically a MSFOL signature and is capable to satisfy some extra requirements.

By using the concepts of signatures, formula and predicates, they have defined some rules for the logical framework. The PEPS is the extension of the MSFOL which works for limited or finite data so their rules are also applicable to the finite data. They also mentioned that the MSFOL formula should be converted to a pseudo-Boolean logic formula to analyse it. Furthermore any compatible solver could be used for this purpose.

The PEPS implementation in the proposed tool is a three steps procedure where grounding operation gives the grounded formula in the first step which is converted to a bit-vector expression using the bit-vector encoding in the second step of this process. In the last step of this procedure, the bit-vector expressions are converted into clauses which are in pseudo-Boolean form and give us the pseudo-Boolean formula.

Using the formulas defined in the proposed logical framework, authors have provided a mechanism to detect the inconsistency, incompleteness, applicability and minimality. It provides the reliable solution because it is based on the logical solvers which themselves are stable. But it is limited to the discrete and limited data without the involvement of contextual attributes in the expressions.

2.5. Matrix-Based Approach

In mathematics, the matrices are usually used for the representation of linear functions and are also used to find the solution for a set of linear equations. In computer science, matrices are commonly used in computer graphics, where they are used to project an image in n-dimensional image in some other m-dimensional co-ordinate system. In the context of access control policy validation, some researchers have used these matrices in collaboration with other tools to find out the problems with access control policies. Some of those methods will be discussed in this section.

Bei *et al.* in [8] have discussed about the existence of many conflict detection algorithms to detect conflicts in ACPs. But according to them, these algorithms are application and policy specification dependent. So these algorithms cannot be reused neither extended to meet some extra requirements.

Authors, in this paper have proposed a solution for this problem and have developed a matrix based algorithm which is independent of application domain. They consider that all kinds of policies like package filter policies, authorization policies and obligation policies belong to ACPs.

Authors have defined the ACP and its different components. The components of a policy rule are modality, event, condition, action, subject and target. These components are called policy field. According to them, to detect a conflict in policies, it is important to define the relativity of their rules. Authors have defined different types of relationships between each policy field. Depending upon these policy fields, six policy field matrices have been created to denote the modality, subject, event, condition, target and action fields of any two rules. Existence of relationship between two rules is denoted by "1" and "0" is used when there is no relation between two fields of different rules. For the purpose of policy rule modelling, another matrix named policy rule matrix is created which is further used to create a policy conflict matrix.

Based upon the matrices created before (relation matrix and conflict matrix) an extensible algorithm (MGCD) has been defined to detect the conflicts. This algorithm has been divided into two phases and it does not describe the policy conflict in the algorithm. Conflict is described in the conflict matrix.

Authors have used the matrix approach to detect the policy conflicts. They claim that their algorithm is extendable and can be applied for different applications but its time complexity is very high when it has to detect conflicts from large number of rules. Its time is directly proportional to the square of the number of rules. i.e.. $t = k\ (number\ of\ rules)^2$.

Huang *et al.* in [14] have addressed RBAC model and have proposed a mechanism to detect conflicts or inconsistencies in access control policies. According to them, it is more complicated task to detect the inconsistencies in this model because of advance constraints supported by this model.

This paper discusses all the elements of the RBAC policy model which includes role hierarchies, separation of duty constraint and cardinality constraints. The authors have presented an inconsistency detection algorithm which includes the above mentioned elements of the RBAC policy model and based on another algorithm (Tarjan's SCC algorithm [38]) mentioned in the paper.

According to the authors, RBAC policy is a 7 tuple rule which includes (U, R, P, RH, RP, UR, C) which represents user, role, permission, role hierarchy, role permission, user role and constraints respectively. In this paper they have discussed static constraints only and discussion of dynamic constraints is beyond the scope of this paper.

Mainly they have focused on separation of duty (SOD) technique and have discussed three types of SOD in RBAC, which are permission separation SOD-P, role

separation SOD-R and user separation SOD-U. Furthermore two types of cardinality constraints also have been discussed which include cardinality constraint on permissions (CC-P) and on the role (CC-R). All these are the important part of the author's inconsistency detection algorithm.

The authors have presented a seven-step mechanism which is followed while developing an access control system using the RBAC model. RBAC Policy is the core component of this model. They have presented the concept of Boolean matrices which further have been used in their proposed algorithm. They also have discussed six types of inconsistency problems which are: inconsistency between RH, inconsistency between RH and SOD-R, inconsistency between RP and SOD-P, inconsistency between UR and SOD-R, inconsistency between UR and CD-R and inconsistency between RP and CD-P. Their algorithm is based upon the Tarjan's algorithm which uses the concept of strong connected components (SCC) in role graph (RG) and based on the DFS algorithm. Time complexity of the algorithm is $O(lm^2)$ where $l = |U|$ and $m = |R|$.

2.6. Mutation Testing Approach

Mutation testing is a testing approach and is used for the software testing. In this technique the code of the existing program is modified in some ways to produce different output of the original program. The modified versions of the original programs are called mutants and their output is compared with the output of the original program. If the two outputs are different, then the mutant is said to be killed and the original output is tested against the other mutant. Higher mutant killing percentage represents the high reliability of the original program. In access control

policy validation case, some researchers have used this technique for the validation purpose. In this section, we will discuss those methods.

E. Martin and T. Xie in [19] have presented a framework to detect the faults in the ACPs which includes a fault model for automated mutation testing of access control policies and it also includes the mutation operators used for this fault model, evaluates the coverage criteria for test generation and selection and also describes the relationship between the structural coverage and effectiveness of fault-detection. Furthermore a tool Margrave [30] has been used for the verification of access control policies which also performs the change-impact analysis on two versions of a policy to reveal the semantic differences between them.

The authors have applied the software testing techniques to detect the defects in the access control policies. In software testing test inputs are passed to the software program to generate test outputs and which are compared with the original outputs. Similarly test requests are passed to the Policy Decision Point (PDP) and the returned responses are compared with the expected responses for verification.

In this work they have used previously defined policy coverage criteria and also a policy coverage measurement tool to know the quality of tests performed on the policies.

Five elements of the XACML policies have been considered for mutant generation, which are: PolicySet, Policy, Rule, Target and Condition. Different combining algorithms to combine different decisions into one decision have been used, e.g. first-applicable, deny-overrides, permit-overrides and only-one-applicable.

Policy coverage, rule coverage and condition coverage are the three types of policy structural coverage used for coverage measurement. Previously developed tool has been used for the random test generation and different tools like Cirg which uses

Margrave have been used for random test generation. To select the reasonable number of tests generated by the test generators, the idea of the minimal representative set has been used.

Different mutation operators defined in this framework have also been discussed in details which are used to generate mutant policies for a given policy. Techniques to detect the equal mutants have also been discussed.

An experiment has been conducted on different policies by using three types of request sets: Cirg based change-impact analysis, randomly generated and subset of randomly generated. The Cirg was supposed to be a good one by killing 59% of mutants.

This framework discusses the general faults present in the policies defined using XACML and it does not focus in depth on inconsistency and incompleteness issue.

E. Martin [6] has discussed the mechanism for effective testing of ACPs. Testing procedure has been divided into three phases where the first phase is named as fault model and mutation testing, second phase deals with the criteria for structural coverage and third phase is the test generation.

Fault models have been used to improve different testing techniques for ACPs and their effectiveness against different faults. Faults have been divided into two categories. i) Semantic faults which are considered as logical faults in ACPs. These faults may present in condition functions, policy generation algorithms and policy evaluation order and may not be detected during static analysis. ii) Syntactic faults which lead to syntactically incorrect policies and can easily be detected. Author aims to develop a policy editing tool to detect and log the faults. It will help to improve policy language design and tools and will reduce fault occurrences.

Structural coverage is further divided into basic coverage criteria and improved coverage criteria. For basic coverage criteria, it is ensured that maximum number of rules, policies, conditions etc. should be tested to test different kind of faults. For this purpose, at least one request should be generated that includes a large number of rules. Policy, rules for a policy and conditions for a rule are three main entities to be considered for testing. In case of improved coverage criteria, policy and rule combination and their ordering is also considered for testing. To test the effectiveness of these coverage criteria, a prototype has been implemented by the author. This prototype shows the less number of requests and relatively low loss in fault detection capability in case of basic coverage criteria and even lower loss in fault detection capability is expected in improved coverage criteria.

Three different techniques have been used in test generation phase. These techniques are i) random test generation, ii) test generation based on solving single-rule constraints and iii) test generation based on solving multiple-rule constraints. In case of random test generation requests in a policy under consideration are randomly generated from the set of requests in that policy. To generate tests based on basic coverage criteria, a rule in a policy and all constrains are tested in ii. In the third technique specific tests are generated to satisfy the improved coverage criteria.

This paper deals with the criteria to test ACPs for fault prevention. It does not provide a solution to remove faults found during this process. It discusses the general faults in the ACPs whether static or logical but gives no idea about inconsistency and incompleteness problems.

Xu *et al.* in [18] have proposed a model based approach to test the access control policies for incompleteness problem. It supports the automated testing and test sets are generated by integrating the access control rules and conditions associated

with the activities. A test automation framework has been used for the test code in various languages like Java, C, C++, C# and HTML/Selenium IDE, but in that paper two java based systems have been used as the test cases.

The authors in this work have followed the software testing approach where test cases are generated for the testing of software to find errors. Similarly, in this model test cases are generated for individual access control rules to detect the incompleteness in those rules. It uses the models of the software under test (SUT) to generate test cases. The proposed model generates executable access control tests from the specifications of the model-implementation description (MID). MID specification consists of model-implementation mapping description. The proposed model has been implemented using MISTA (formerly known as ISTA) framework [25] [26] which automatically generated the test code in many languages mentioned above. It is represented by a Predicate/Transition (PrT) net. It is constructed from the access control rules and functional requirements of the SUT. In addition to this, mutation analysis of access control implementation has been applied to test the fault detection capability of the proposed model. Mutants are created by using the MutaX tool by using faulty rules and as a result of test execution; they are killed if a failure is reported by the system.

The access control rule defined and used for this model is a five tuple which consists of role/subject, object, action/activity, context which represents the Boolean expression and a set of authorization types. Three types of authorization types have been used which include: Permission, Prohibition and Undefined.

To analyze and debug the specifications of the test models constructed using PrT nets [23] [24] [25], three approaches are used: verification of transition reachability, verification of state reachability and model simulation. In the proposed

model test cases are generated from the test models. MISTA supports automated test generation for different coverage criteria like reachability tree coverage, state coverage and transition coverage. It also provides partial ordering and pairwise combination technique to reduce the number of tests generated.

According to the authors, the proposed model can efficiently detect and resolve the incompleteness problem in access control policies but it does not address the inconsistency or redundancy problems. Due to the large number of test cases, it is not feasible to use this model for large programs but it can be used by dividing the large system into smaller components or modules.

2.7. Other Techniques

Shafiq *et al.* in [20] have addressed the event-driven access control policies and have proposed a framework to detect and resolve the inconsistencies in those policies. An integer programming approached has been used by them for the detection and resolution of inconsistencies.

Two types of hierarchies have been used in the RBAC model which are: inheritance hierarchy and activation hierarchy. A separation of duty (SoD) constraints is also the main part of the RBAC model and Role-specific SoD and User-specific SoD are the basic constraints used for this purpose. SoD constraints identified in this paper also have been composed from these constraints. Furthermore two types of dependency constraints have been defined to show the relations between nodes in the type graph used by the authors: strong dependency and weak dependency. Users, roles and permissions have been represented as nodes in the graph and the edges represent the association and constraints between different nodes.

Integer programming (IP) technique has been used to detect and resolve inconsistencies. For this purpose IP constraint transformation rules have been defined. For users, these rules have been divided into four main categories: Hierarchy and assignment, role enabling, SoD, Dependency triggers. The idea of proxy users has also been used and active proxy and passive proxy are the two terms used for the proxy users. After all an algorithm has been developed that takes an event-driven policy graph as the input and returns the consistent and fault-free graph.

Jin-hua *et al.* in [17] have presented a policy-based firewall management framework to manage different kind of firewalls. In this framework it also provides a mechanism to detect inconsistencies in the rules defined by the administrators. The approach used in this paper is based on the IETF policy framework and it can manage hybrid firewalls and application layer firewalls.

The architecture of this framework consists of the four main components which are: Policy Repository (PR), Policy Management Tool (PMT), Policy Decision Point (PDP) and Policy Enforcement Point (PEP). It also includes Policy Analyze tool and a Monitor and Post-test Analyze tool. It also includes an Enforcement Validation Engine. From these components the Policy Analyze tool analyzes policies for inconsistency problems and provides a mechanism to detect the inconsistencies in the policies defined by the administrators. Each rule in this framework consists of six attributes which include: protocol, IP addresses and port of both sender and receiver and the action upon the acceptance or rejection of packets from the firewall. Inconsistency problems have been classified as the shadowing problem, correlation problem, generalization problem and redundancy problem based on the relations between different rules.

A GUI based tool has been developed using Java which implements different inter and intra firewall inconsistency detection algorithms.

Stepien *et al.* in [12] have discussed different strategies which are helpful to avoid the risks of inconsistencies. This is a general discussion and does not provide any algorithm or specific technique to eliminate the inconsistencies from the access control policies. It shows that how can we use the modern languages, tools and techniques while writing these policies to avoid inconsistencies. It also discusses about the auditing techniques to detect inconsistencies at compile time and run time. The ways to improve the efficiency of the systems when a large number of rules is used to ensure restricted access to resources have also been discussed.

First of all, current methods for conflict detection in rule based policies, especially in the context of XACML have been reviewed. Then the need for a user friendly non-technical notation and interface to define and verify the policies has been discussed. According to the authors, such a notation makes it possible to easily use complex expressions in the condition part of the rules and without such complex conditions the equivalent 'simple' rule sets get large and difficult to build and explain. These complex conditions in XACML lead to more compact rule sets which can be built and understood by policymakers themselves without relying on specialized IT personnel. At the end they have demonstrated how the use of complex conditions leads to a very efficient implementation which encodes the rules in Prolog and combined with the backtracking mechanisms of Prolog. This results in a very efficient method of checking the rule sets for inconsistencies.

Authors have emphasized in this work that the use of complex conditions in rules leads to compact rule sets and instead of writing many simple rules to satisfy one condition, rules can be derived with the complex expressions to replace those multiple

simple rules. This can be achieved by using the new ACP languages like XACML and use of GUIs is also helpful to achieve this goal.

There are some steps need to be taken to reduce the risks of inconsistencies. Use of non-technical notations and related tools like GUI is one of those steps. Then instead of using simple rules containing only one condition, rules with complex conditions may be used which in result combines several rules in on single rule. Now static modal conflict detection strategies can be used which can detect the inconsistencies on both compile time and run time. Modal conflict detection techniques will also be helpful at this stage to detect the inconsistencies by auditing. For auditing different queries will be written to get the policies and by examining those resulting policies, inconsistencies can easily be detected. At the end the scalability and performance issues can also be solved using complex conditions and compact rule sets.

In [21], Tekbacak *et al.* have proposed a framework to ensure the security of the multi agent systems (MAS) using the XACML based access control policies. In this framework the semantic structure of MAS has been used with the XACML characteristics. XACML and OWL have been used in the data layer and have modified to description logic (DL) concepts. Furthermore the combination of agent domain ontology and agent security ontology has been used with the XACML policy set.

Agents, reference monitor, agent domain ontology, agent main security ontology and policy ontology are the main components of the proposed MAS architecture. XACML ontology translation to the DL is also a main component of the system which includes a policy warehouse where policies are stored. Furthermore XACML framework used in this system also consists of three components: Policy enforcement point (PEP), policy decision point (PDP) and policy administration point

(PAP). All these components play an important role to define and enforce the consistent security policies.

This paper does not directly deal with the problem of inconsistency or incompleteness but it implements the XACML framework for MAS which itself tries to make them consistent and complete by using its own components.

2.8. Trend Analysis

In this section we have analyzed the existing techniques according to the approach used and the areas covered by researchers for validation of inconsistencies in access control policies.

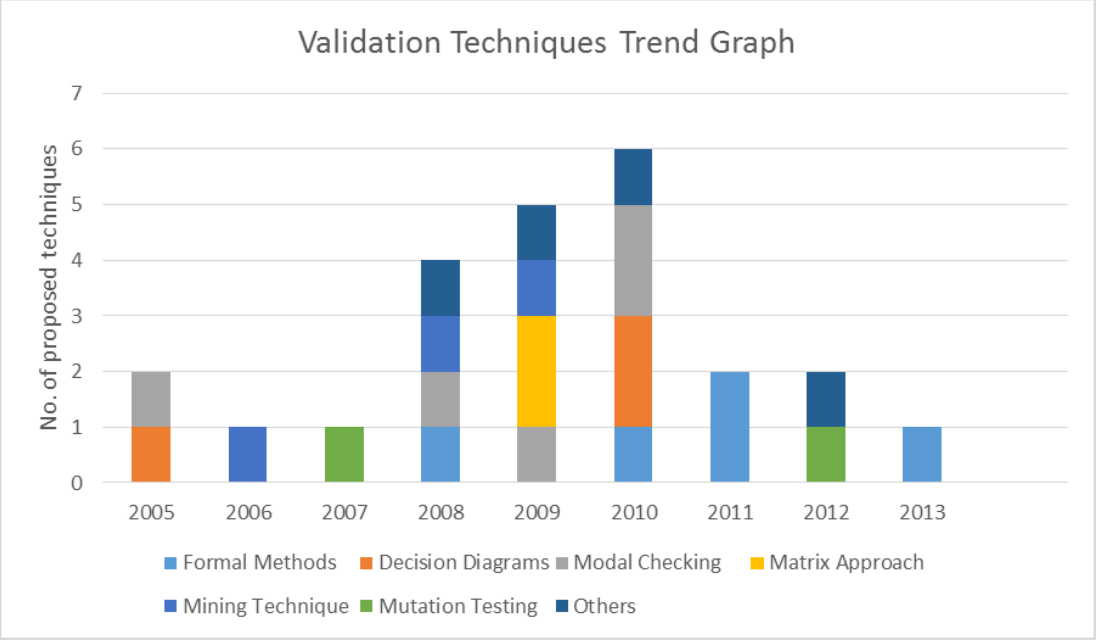


Figure 2.2. Graph showing the ratio of validation methods adopted by researchers from 2005-2013

We analyzed the proposed techniques according to their effectiveness in handling different kind of above mentioned problems and attributes, we have used for

their comparison. Figure 2.2 shows a trend graph for different proposed techniques during 2005-2013, which we are classified in different categories. It is clear from the graph that most of the researchers have used formal methods and modal checking approaches to validate the access control policies.

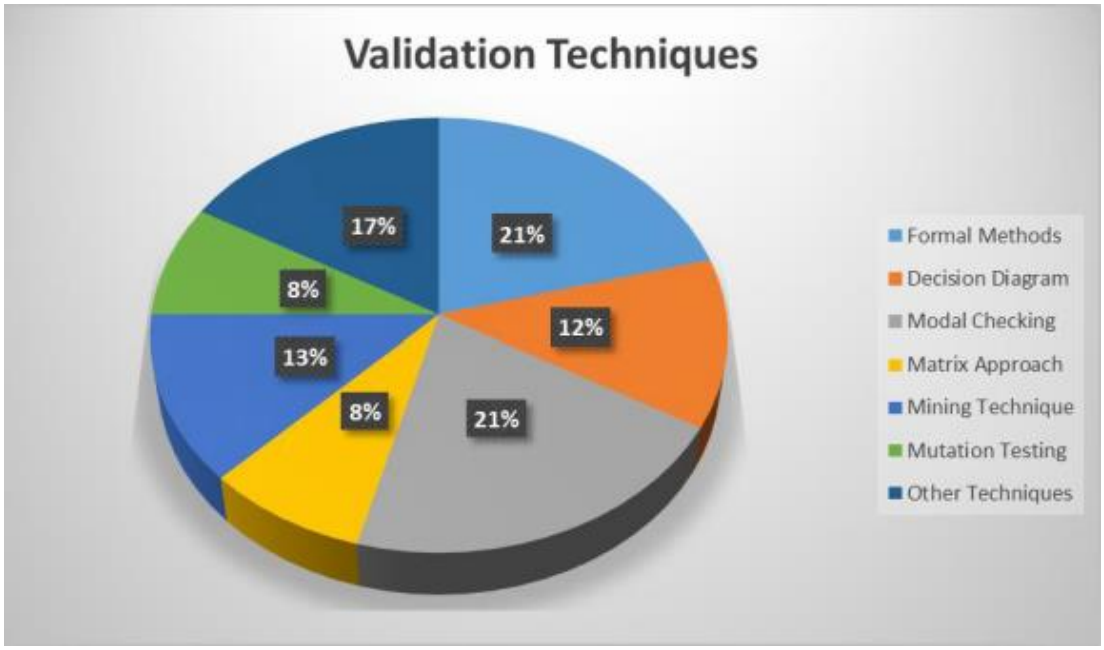


Figure 2.3. The percentage distribution of different types of proposed validation techniques

Chart given above (Figure 2.3.) shows the percentage distribution of the techniques used for the comparison purpose. It gives us a clear picture by showing the percentage of each individual technique used by the researchers. Formal Methods and Modal Checking techniques have the highest percentage of 21% each whereas the Matrix based and Mutation testing approaches both have a contribution of 8% each. Furthermore, 17% of them have used their own techniques for this purpose.

In the following figure (Figure 2.4.) we have given a percentage distribution of the properties to show that how the researchers have addressed these issues in their proposed techniques.

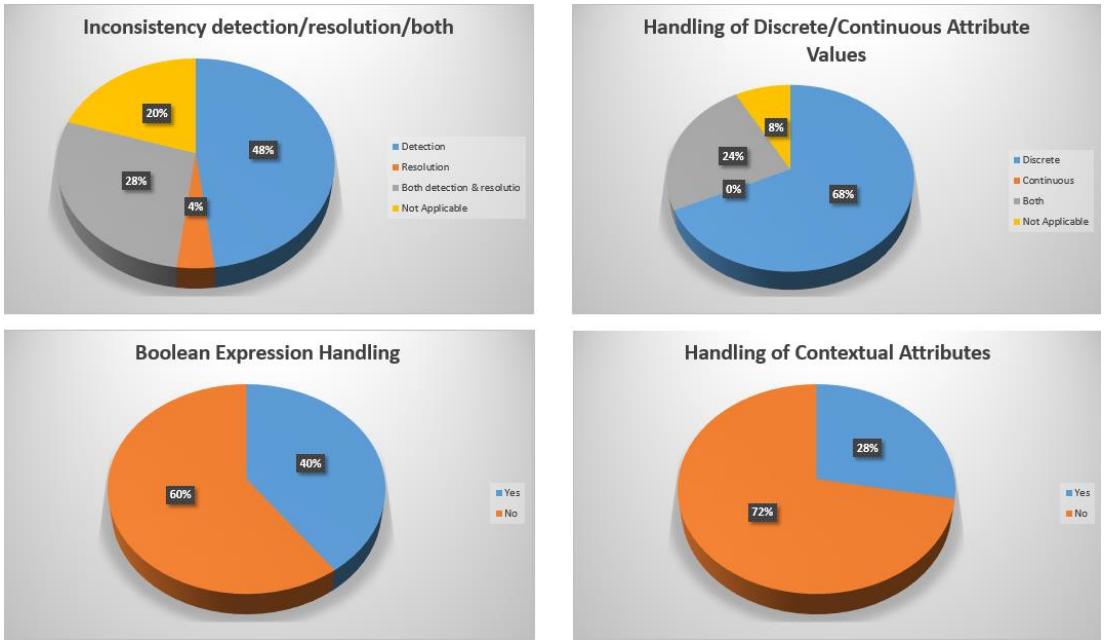


Figure 2.4. Percentages of issues addressed in compared techniques

Chapter 3

Proposed Solution

3.1. Inconsistency and Related Concepts

As discussed above, inconsistency in the policy set exists when any two rules in that policy set lead to the contradictory outcomes. For example, if a rule defined in the policy set allows a user to access some resources during a specific time span but there exists some other rule in the same policy set which deny the user to access the same resource during some other time slots. However, if these time slots are same or they overlap, then we say that these two rules lead towards the contradictory statements and therefore they are not consistent. Hence, the policy set is said to be inconsistent. The rules defined by the administrators consist of different attribute values and the values of these attributes lead them to some decision based upon these attribute values. In the following section, we will discuss in detail about these attributes.

3.2. What Is Inconsistency?

To define a rule in a policy set, various attributes are used to define different entities like user, resources, action, context, category or decision etc. Among all these attributes, the decision attribute define the class to which the specific rule belongs. There may be different classes like permit, deny and undefined. These classes define the kind of permission granted to the user, e.g. access granted to a specific user to access specific resources under certain conditions or revoked or it is undefined etc.

Let $S = \{s_{1,} s_2\, s_{3,} \ldots., s_n\}$ $n \in N$, $O = \{o_1, o_2, o_3, \ldots, o_m\}$ $m \in N$, $C = \{c_1, c_2, c_3, \ldots, c_l\}$ $l \in N$ and $A = \{a_1, a_2, a_3, \ldots, a_k\}$ $k \in N$ are the sets of subjects, objects, contexts and actions containing *n* subjects, *m* objects, *l* context values and *k* actions respectively and let $D = \{permit, deny, undefined\}$ be the set of category/decision attributes. An access control policy is considered to be a four tuple rule $(s, o, a, c) \rightarrow d$ where $s \in S, o \in O, a \in A, c \in C$ and $d \in D$. If R is the set of rules, then for any two rules r_i and $r_j \in R$ such that $i \neq j$, if r_i and r_j lead to the contradictory decisions i.e. $r_i \rightarrow d_x$ and $r_j \rightarrow d_y$, $x \neq y$ then the policy set is said to be inconsistent.

3.3. Example of Inconsistency

Let us consider the example of two employees (Manager and Cashier) working in a bank and they need to access some records to perform different tasks. Only the Manager has the right to perform any kind of operation on the customer's records where the Cashier can only view the customer details to perform some transactions. The bank administration has reserved two days (Monday and Tuesday) to open new

accounts. In case of any change in customer information, they can visit the bank on Wednesday and Thursday. Friday is the last working day of the week, the management will review the records of the customers and that day they can delete/block the account of inactive customer accounts.

Table 3.1. Access rights defined for different roles to perform different operations on resources

Rule	Subject	Object	Action	Decision	Day
1	Manager	Record File	View customer info	Permitted	Mon-Fri
2	Manager	Record File	Add new customer	Permitted	Mon, Tue
3	Manager	Record File	Update customer info	Permitted	Wed-Thu
4	Manager	Record File	Delete customer record	Permitted	Fri
5	Cashier	Record File	View customer info	Permitted	Mon-Fri
6	Cashier	Record File	Add new customer	Denied	Mon-Fri
7	Cashier	Record File	Update customer info	Denied	Mon-Fri
8	Cashier	Record File	Delete customer record	Denied	Mon-Fri

Table 3.1, shows the various rules defined to perform different operations on the record file by different users. It is clear that both Manager and Cashier can view the records in that file throughout the week, but only Manager can add new customers in record file. In addition, he can update the customer information and can also perform the delete operation on inactive accounts. It is clear from the above-mentioned rules that there is no inconsistency. But due to some reasons, suppose the Manager delegates his delete record rights to the cashier. Then the rule 9 is needed to add in the rule set.

Table 3.2. New rights assigned to cashier if manage delegates the delete right to him

Rule	Subject	Object	Action	Decision	Day
5	Cashier	Record File	View customer info	Permitted	Mon-Fri
6	Cashier	Record File	Add new customer	Denied	Mon-Fri
7	Cashier	Record File	Update customer info	Denied	Mon-Fri
8	Cashier	Record File	Delete customer record	Denied	Mon-Fri
9	Cashier	Record File	Delete customer record	Permitted	Fri

Now according to the new rules defined in Table 3.2, Cashier is allowed to delete customer records on Friday, which contradicts with the rule 8, which states that Cashier cannot perform delete operation on customer records. This shows that the rules defined in this policy are inconsistent.

3.4. Inconsistency Detection Algorithm

In this section, we will discuss the proposed algorithm for the detection of inconsistencies in access control policies. This algorithm takes the access control policies in the form of a decision tree. As discussed above, the rule is defined in the form of four tuple, which includes subject, object, action and context i.e. $(s, o, a, c) \rightarrow d$. The validation process in this algorithm is completed in two phases. In the first part, algorithm takes a decision tree as an input and divides it into sub-trees based upon the number of decision attribute values. In the second phase, algorithm takes sub-trees as an input and compares them recursively to detect inconsistencies.

3.4.1. Decision Tree Hierarchy

In the tree, the Decision attributes (*d*) are on the top of the hierarchy that are the children of the root node as shown in Figure 3.1. These nodes include the action attributes in their child attribute list so the action attributes are on the second level in the tree hierarchy. Object attributes are the direct children of the action attributes and exist in the children attribute list of the action attributes. So they are on the third level in this hierarchy. In this tree hierarchy, the subject nodes are on the fourth level and they exist in the children attribute list of the objects which are the parents of subject attribute nodes. Subject attribute nodes in turn contain the contextual attributes in their children attribute lists and exists on the fifth level of this hierarchy and they are also the leaf nodes of the policy tree. It then starts the validation process and to detect the inconsistencies and returns the inconsistent rules in case inconsistencies found in the policies.

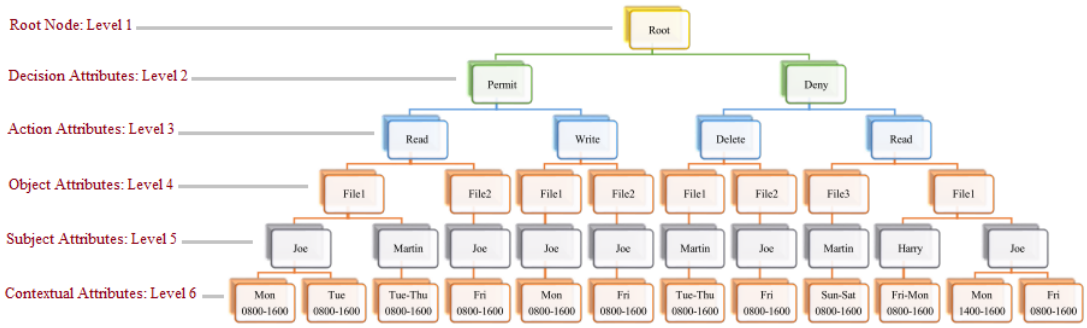
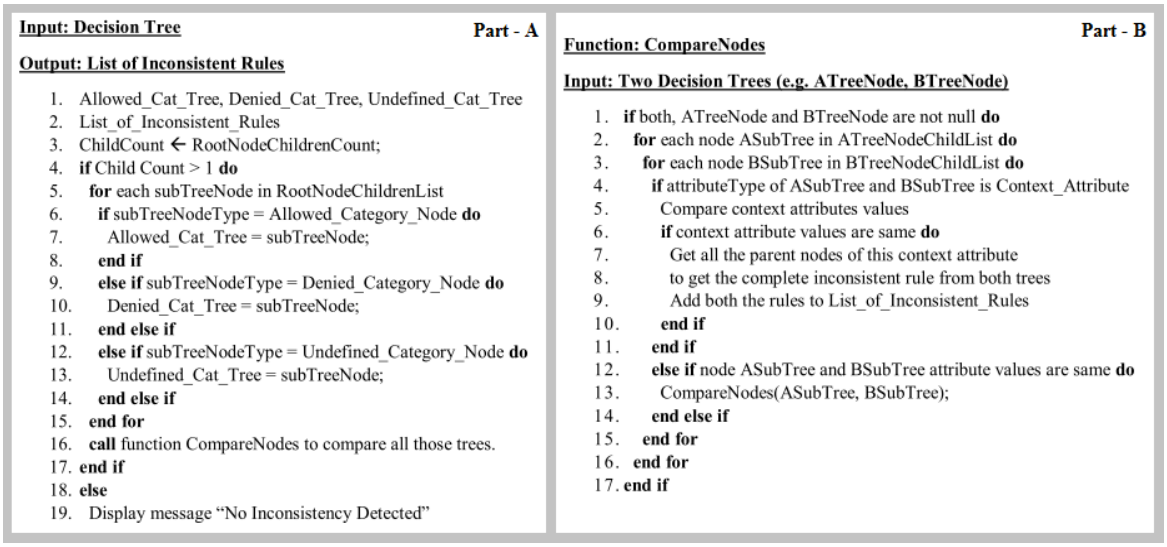


Figure 3.1. Sample hierarchy of the decision tree

3.4.2. Inconsistency Detection Process

As discussed above, the proposed algorithm consists of two parts that are clearly shown in Figure 3.2. In the following paragraphs, we will briefly describe the working of this algorithm.



Input: Decision Tree — **Part - A**

Output: List of Inconsistent Rules

```
1.  Allowed_Cat_Tree, Denied_Cat_Tree, Undefined_Cat_Tree
2.  List_of_Inconsistent_Rules
3.  ChildCount ← RootNodeChildrenCount;
4.  if Child Count > 1 do
5.    for each subTreeNode in RootNodeChildrenList
6.      if subTreeNodeType = Allowed_Category_Node do
7.        Allowed_Cat_Tree = subTreeNode;
8.      end if
9.      else if subTreeNodeType = Denied_Category_Node do
10.       Denied_Cat_Tree = subTreeNode;
11.     end else if
12.     else if subTreeNodeType = Undefined_Category_Node do
13.       Undefined_Cat_Tree = subTreeNode;
14.     end else if
15.   end for
16.   call function CompareNodes to compare all those trees.
17. end if
18. else
19.   Display message "No Inconsistency Detected"
```

Function: CompareNodes — **Part - B**

Input: Two Decision Trees (e.g. ATreeNode, BTreeNode)

```
1.  if both, ATreeNode and BTreeNode are not null do
2.    for each node ASubTree in ATreeNodeChildList do
3.      for each node BSubTree in BTreeNodeChildList do
4.        if attributeType of ASubTree and BSubTree is Context_Attribute
5.          Compare context attributes values
6.          if context attribute values are same do
7.            Get all the parent nodes of this context attribute
8.            to get the complete inconsistent rule from both trees
9.            Add both the rules to List_of_Inconsistent_Rules
10.         end if
11.       end if
12.       else if node ASubTree and BSubTree attribute values are same do
13.         CompareNodes(ASubTree, BSubTree);
14.       end else if
15.     end for
16.   end for
17. end if
```

Figure 3.2. Proposed algorithm to detect inconsistencies in access control policies

Step 1:

In this step, the main tree will be divided into the sub-trees equal to the number of decision attributes. For this purpose it will count the number of decision attribute nodes that are the children of the root node (Part A, Line: 3). If there is only one decision attribute node in the children node list of the root node (Part A, Line: 4), then the algorithm will stop and it will display no inconsistency found message (Part A, Lines: 18, 19). In another case, the main tree is divided into the sub-trees equal to the number of category attributes in the children attributes list of the root node (Part A, Lines: 5-15). Suppose there are two category attributes, *permit* and *deny* as shown in a sample hierarchy tree in Figure 3.1, then in that case the main tree will be divided into the two sub-trees where all the policies with category attribute value "*permit*" will be present in the first tree having same category attribute value as the root node of the tree. Similarly, all the other rules will be present in the other tree with category attribute

value "*deny*" as the root node. Resulting sub-trees with decision attribute as root nodes are shown in the Figure 3.3.

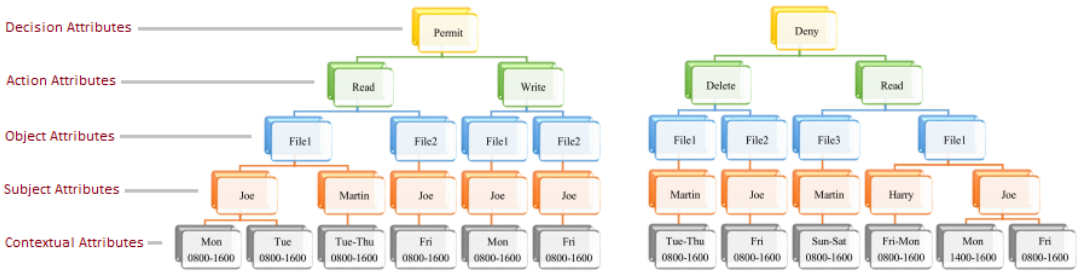


Figure 3.3. Sub-trees generated with decision attribute as the root node.

Step 2:

After having separate trees for each decision node as shown in the Figure 3.3, our algorithm will start comparing two sub-trees using the *CompareNodes* function (Part A, Line: 16). It will compare only if both of the trees are not null (Part B, Line: 1). After that it will get the child nodes of the first tree and will start comparing it with the child nodes of the second tree (Part B, Lines: 2, 3). If the child node type in both trees is action and the node values are also same, it will pick those nodes and will call the *CompareNodes* function again (Part B, Lines: 12-14). In Figure 3.3, the child node of decision attribute node is action node and its value "Read" is same in both sub-trees. Now the action node will become the root node of both the trees passed to the *CompareNodes* function as shown in Figure 3.4.

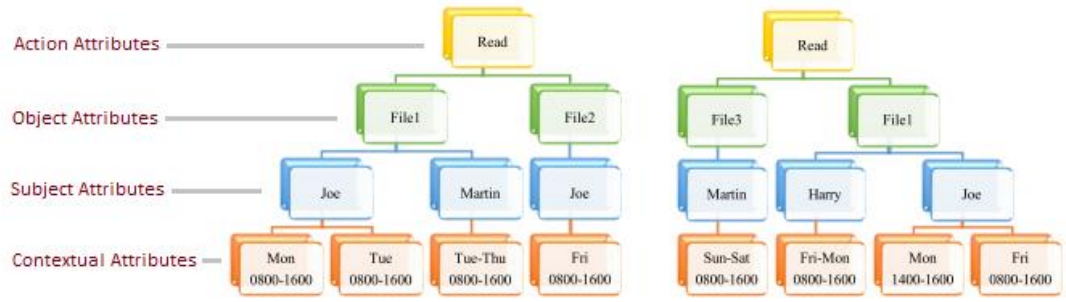


Figure 3.4. Sub-trees generated with action attribute as the root node.

Again as both the trees shown in Figure 3.4 are not null (Part B, Line: 1), it will get the child nodes of the root node (action node is root node here) and the object attribute nodes are the child nodes at this step (Part B, Lines: 2, 3). Now it will compare the values of object attributes and will call the *CompareNodes* function again if they have the same values in both trees (Part B, Lines: 12-14). As shown in the Figure 3.4, object nodes having "*File1*" are same in both the trees so now sub-trees will be having them as root nodes. The Figure 3.5 shows the resulting trees passed to the *CompareNodes* function in result of this comparison.

Figure 3.5. Sub-trees generates with object node as the root node.

The *CompareNodes* function will compare the trees shown in Figure 3.5 where object attribute node is the root node. It is clear that the child node type is subject node and "*Joe*" is the same attribute value in both the trees. So *CompareNodes* function will be called again and this time the subject attribute node will be the root node in both the sub-trees passed as parameters. The Figure 3.6 shows the resulting sub-trees with subject attribute nodes as the root nodes.

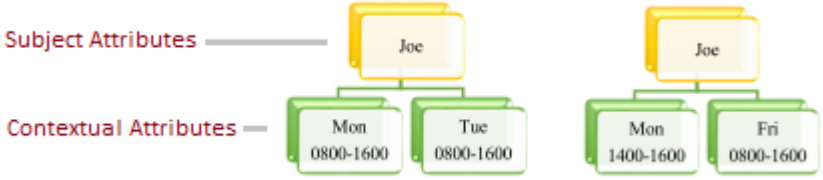


Figure 3.6. Sub-trees generated with subject node as the root node.

These trees will be passed to the *CompareNodes* function and they have contextual attributes as their child nodes. So this time the *CompareNodes* function will not be called again and contextual attributes will be compared in step 3 of the algorithm.

Step 3:

As mentioned above, if the child node type in both the trees is context node, the *CompareNodes* function will not be called because these are the leaf nodes of the decision tree. It also indicates that all the other attributes are same. Now, it will start comparing the contextual attribute values (Part B, Lines: 4, 5). If the contextual attributes have the same values, it means both these rules are same. In Figure 3.6, we can see that there is a contradiction in time attribute. The user is permitted to access the resource on Monday from 0800 to 1600 but on the same day, he cannot access the resource from 1400 to 1600. So it will get all the parent nodes of those contextual attributes to get those rules (Part B, Lines: 6-8) as shown in Figure 3.7. Here all attribute-values of both the rules are same, it means they are inconsistent and hence they will be stored in the list of inconsistent rules (Part B, Line: 9). The Same process will be repeated until all the sub-trees generated during step 1 are compared with each other.

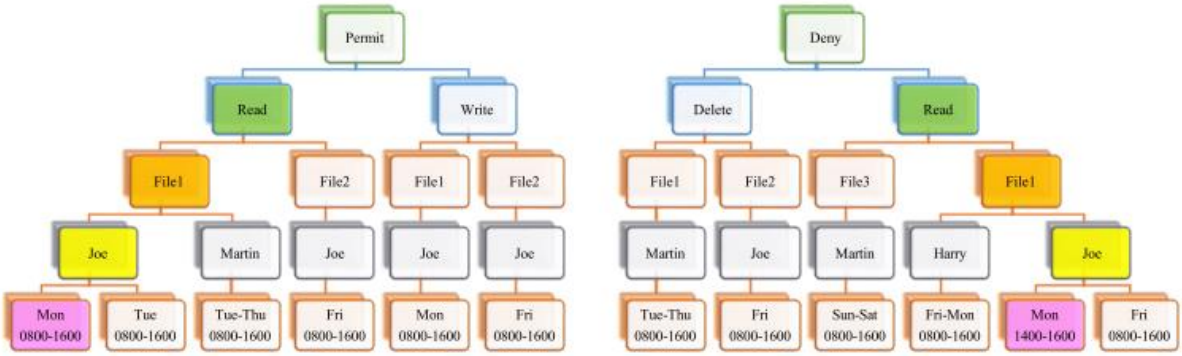


Figure 3.7. Rules with contradictory decisions identified.

Chapter 4

Proposed Method Implementation

4.1. ACPs Validation Suite

We have implemented our proposed algorithm and have developed a tool named “APCs Validation Suite”, which takes the access control policies, defined in XML. This simple program takes an XML file as input and displays the rules defined in XML file. By implementing the proposed algorithm, it performs the validation process and displays the inconsistent rules along with their Ids. In Figure 4.1, we have shown an XML file that contains twenty-three rules to access different resources by different users. As we already have mentioned that we have considered the four-tuple rules for these policies. Each rule is defined as an element in the XML file and the attributes of this element represent the attributes of policy rules. The id attribute defines the rule’s identity, the subject attribute represents the users who want to access

the resources, object attribute represents the resources accessed by different user and the action they have to perform on those objects is represented by the action attribute value. In this example, we have seven subject values, user1 till user7. Object attribute also have described eight resources like File1, File2 etc. Three operations, Read, Write and Delete have defined in the action attribute value. The rest of the attributes are the contextual attributes, which define time period, months and the age group of the users. Figure 4.2 shows the APCs Validation Suite, where the upper half of the screen shows the contents of the selected file and the lower half shows the rules with contradictory decisions.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
    <Rule id="Rule1" subject="user1" object="File1" action="Read" startTime="08:00" endTime="16:00" month="Apr-Jul" age="&gt;=30" category="Allowed"/>
    <Rule id="Rule2" subject="user1" object="File1" action="Read" startTime="08:00" endTime="16:00" month="Jan,Aug" age="&gt;50" category="Denied"/>
    <Rule id="Rule3" subject="user2" object="File1" action="Read" startTime="08:00" endTime="16:00" month="Jan,Aug,Nov" age="&lt;60" category="Allowed"/>
    <Rule id="Rule4" subject="user1" object="File1" action="Write" startTime="08:00" endTime="16:00" month="Dec" age="40" category="Allowed"/>
    <Rule id="Rule5" subject="user2" object="File1" action="Write" startTime="08:00" endTime="16:00" month="Jan-Dec" age="&gt;=30" category="Allowed"/>
    <Rule id="Rule6" subject="user2" object="File1" action="Delete" startTime="08:00" endTime="16:00" month="Oct" age="&lt;=55" category="Denied"/>
    <Rule id="Rule7" subject="user2" object="File2" action="Read" startTime="12:00" endTime="14:00" month="Sep-Dec" age="50" category="Denied"/>
    <Rule id="Rule8" subject="user1" object="File1" action="Read" startTime="12:00" endTime="20:00" month="Jan-May" age="&lt;=45" category="Denied"/>
    <Rule id="Rule9" subject="user1" object="File1" action="Read" startTime="08:00" endTime="16:00" month="Jan-Dec" age="60" category="Allowed"/>
    <Rule id="Rule10" subject="user1" object="File1" action="Read" startTime="12:00" endTime="20:00" month="Feb" age=">=50" category="Denied"/>
    <Rule id="Rule11" subject="user3" object="File3" action="Read" startTime="08:00" endTime="16:00" month="Apr-Jul" age="&gt;=30" category="Allowed"/>
    <Rule id="Rule12" subject="user3" object="File3" action="Read" startTime="08:00" endTime="16:00" month="Jan,Aug" age="&gt;50" category="Denied"/>
    <Rule id="Rule13" subject="user2" object="File3" action="Read" startTime="08:00" endTime="16:00" month="Jan,Aug,Nov" age="&lt;60" category="Allowed"/>
    <Rule id="Rule14" subject="user6" object="File3" action="Write" startTime="08:00" endTime="16:00" month="Dec" age="40" category="Allowed"/>
    <Rule id="Rule15" subject="user7" object="File3" action="Write" startTime="08:00" endTime="16:00" month="Jan-Dec" age="&gt;=30" category="Allowed"/>
    <Rule id="Rule16" subject="user4" object="File3" action="Delete" startTime="08:00" endTime="16:00" month="Oct" age="&lt;=55" category="Denied"/>
    <Rule id="Rule17" subject="user4" object="File4" action="Read" startTime="12:00" endTime="14:00" month="Sep-Dec" age="50" category="Denied"/>
    <Rule id="Rule18" subject="user5" object="File4" action="Read" startTime="12:00" endTime="20:00" month="Jan-May" age="&lt;=45" category="Denied"/>
    <Rule id="Rule19" subject="user7" object="File6" action="Read" startTime="08:00" endTime="16:00" month="Jan-Dec" age="60" category="Allowed"/>
    <Rule id="Rule20" subject="user5" object="File8" action="Read" startTime="12:00" endTime="20:00" month="Feb" age=">=50" category="Denied"/>
    <Rule id="Rule21" subject="user5" object="File14" action="Read" startTime="12:00" endTime="20:00" month="Jan-May" age="&lt;=45" category="Denied"/>
    <Rule id="Rule22" subject="user7" object="File8" action="Read" startTime="08:00" endTime="16:00" month="Jan-Dec" age="60" category="Allowed"/>
    <Rule id="Rule23" subject="user5" object="File9" action="Read" startTime="12:00" endTime="20:00" month="Feb" age=">=50" category="Denied"/>
</root>
```

Figure 4.1. Sample XML file to detect inconsistencies in access control policies

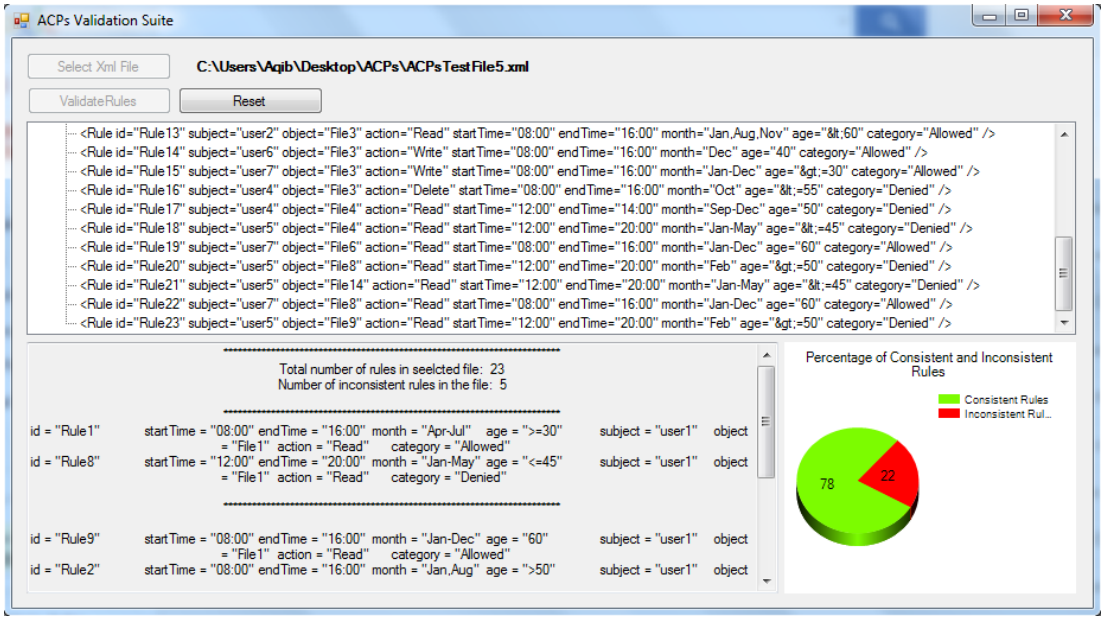


Figure 4.2. ACPs Validation Suite

4.2. System Architecture

ACPs Validation Suite has developed to implement the proposed algorithm. Its main screen has three portions. Upper half of its screen is reserved for the input and shows the contents of the XML file containing the rules for the validation purpose. Currently this system takes only the XML files as the input files. Here the rules defined in the XML file are displayed as a tree. The lower half of main screen is reserved for the output generated by the system in response to the given input. This part has been divided into two parts. The left part shows the number of rules defined in the input file and the number of rules found to be inconsistent. In addition, it also shows the inconsistent rules so that the user could easily find the inconsistencies in the defined rules. The right portion of this output section shows the percentage of inconsistent rules as compared to the total number of rules defined in the input file. So this helps the user to estimate the ratio of errors in the defined rules. This may held the

administrators to find out the suitability of the technique used to define the rules. Percentage of the inconsistencies is displayed in graphical form.

4.3. Modules of the ACPs Validation Suite

Validation process in the ACPs Validation Suite has been divided into five main steps. So in development of Validation Suite this five different modules have been developed to accomplish the tasks performed in those five steps. Following are the main steps involved in the validation process. Figure 4.3 shows the modules of the developed system.

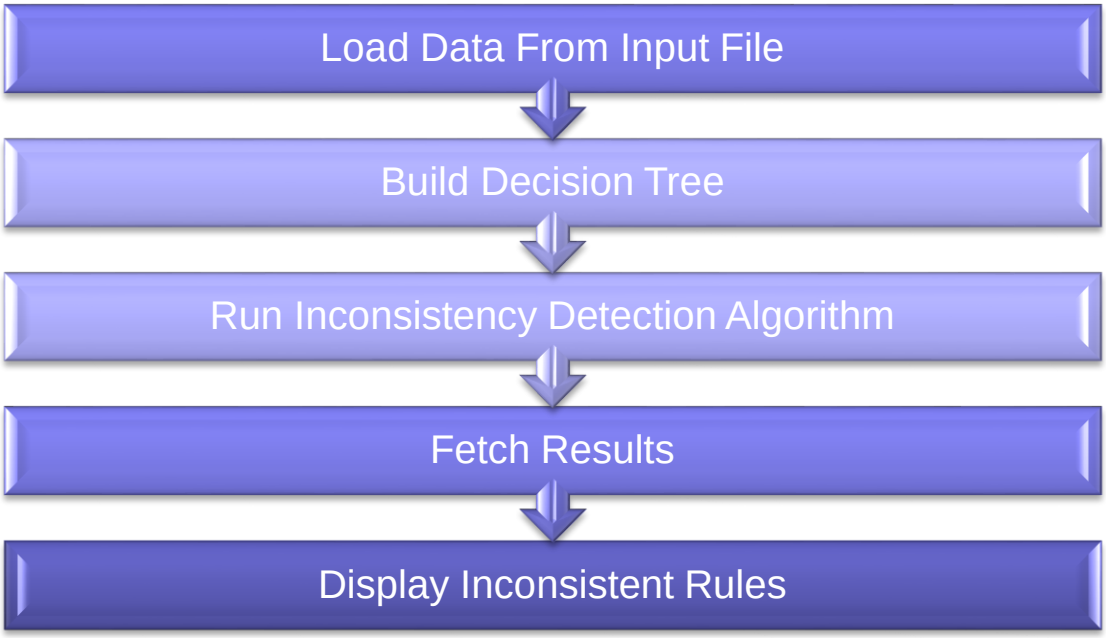


Figure 4.3. Modules of ACPs Validation Suite

4.3.1. Load / Read data from the input file

In this module, the user is supposed to select the file containing the rules defined by the administrator to check the inconsistencies. After selection of the appropriate file, this module reads the data in that file and save this data so that it could be used by the other modules for different purposes. Currently user is allowed to use only XML files for this purpose.

4.3.2. Build decision tree

Data saved in module 1 is then used to build a decision tree because the proposed algorithm takes the input in the form of decision tree. So this tree is built using the rules defined in the selected input file. This tree has its own data structure to hold the tree nodes. Each node mainly has two attributes, type and value. A child nodes list is also associated with each node that contains all child nodes of that specific node.

4.3.3. Application of proposed algorithm

The proposed algorithm takes the rules for validation in the form of decision tree. So after the conversion of rules in this form, the decision tree is passed to the module implementing the proposed algorithm. This module not only implements the proposed algorithm but also contains the methods defined to compare the attribute values especially in case of comparison of contextual attributes. The values of contextual attributes are passed to these methods which then compare those values and the results are returned for further processing.

4.3.4. Collection of results / inconsistent rules

After completion of validation process using proposed algorithm, the system has to check whether inconsistencies have been detected in the rules or not. If there are some inconsistencies, then this module collects the inconsistent rules exist in the policy set so that the administrators could remove those inconsistencies.

4.3.5. Display output

This module generates an output for the user which includes the rules defined in the input file and the inconsistent rules collected in the previous module in case there are inconsistencies in the input file. If no inconsistency is detected, then it simply displays a message declaring that no inconsistencies exist in the policy set. But in both these cases, it displays the actual data defined in the input file. It also displays a graph that shows the percentage of inconsistent rules in the input file.

4.4. Development Tools

The following tools and technologies have been used in development of ACPs Validation Suite.

- C# 4.0 with .Net Framework 4
- Microsoft Visual Studio 2010
- Windows 7 Professional (Operating System)
- Intel Core i5 Processor,
- Hewlett Packard System with 6 GB RAM

Chapter 5

Analysis and Evaluation

5.1. Complexity Analysis of Proposed Algorithm

Complexity of the proposed algorithm depends upon the number of distinct attribute values for different attributes. In the proposed algorithm the main tree is divided into sub-trees according to the number of decision attribute values like permit, deny and undefined. Each sub-tree contains action, object, subject and contextual attribute nodes. Nodes in the tree represent the distinct attribute values for these attributes and hence number of iterations at each level is also dependent to the number of distinct attribute values at that level. So total computational complexity is the sum of complexities on all the levels of the tree. There are two different cases to calculate the complexity at those levels depending upon the number of decision attributes. Let n be the total number of rules defined in the policy set. Let us also consider that a is the number of distinct attribute values for action attribute, o is the number of distinct attribute values for object attribute, s is the number of distinct attribute values for subject attribute and c is the number of distinct attribute values for contextual attribute

values. Formulas to calculate complexity at all these levels have defined below for both cases.

Case 1:

In this case only two decision attribute values are considered, permit and deny. As a result the main tree is divided into two sub-trees, say for example, permit and deny trees.

For Action Attribute: $O(a^2)$

For Object Attribute: $O(o^2 \times a)$

For Subject Attribute: $O(s^2 \times o \times a)$

For Context Attribute: $O\begin{pmatrix} a \times o \times s & if\ c = 1 \\ a \times o \times s \times 3(c-1) & if\ c > 1 \end{pmatrix}$

$n = 2 \times a \times o \times s$

Case 2:

In this case three decision attribute values are considered, permit, deny and undefined. As a result the main tree is divided into three sub-trees, say for example, permit, deny and undefined trees.

For Action Attribute: $O(3 \times a^2)$

For Object Attribute: $O(3 \times o^2 \times a)$

For Subject Attribute: $O(3 \times s^2 \times o \times a)$

For Context Attribute: $O\begin{pmatrix} a \times o \times s \times 3 & if\ c = 1 \\ a \times o \times s \times 9(c-1) & if\ c > 1 \end{pmatrix}$

$n = 3 \times a \times o \times s$

In the Figure 5.1 and 5.2, we have shown the complexity of the proposed algorithm based upon the number of distinct action, object and subject attribute values and the number of contextual attributes in each rule. Table 5.1 and 5.2 shows the complexity for case 1 and case 2 respectively. It shows the number of contextual attributes which range from 1 to 10 and also the number of distinct subject, action and object attribute values which also range from 1 to 10. It is worth mentioning that we have considered the same number of distinct subject, object and action attribute values for comparison purpose. It mean that if a policy set has 2 different subject attribute values then it also contains two distinct object and action attribute values as well. In Figure 5.1, two decision attribute values have considered (case 1) and Figure 5.2 considers the existence of three decision attributes (case 2). From both the graphs, we can conclude that complexity of case 2 is three times higher than the case 1. Also, both graphs show that the complexity increases linearly with the increase in number of contextual attributes whereas it increases more sharply with the increase in number of distinct actions, objects and subjects.

Table 5.1. Complexity analysis with two decision attribute values

		Number of Contextual Attributes									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Number of Subject, Object and Action attribute values	**1**	4	6	9	12	15	18	21	24	27	30
	2	36	52	76	100	124	148	172	196	220	244
	3	144	198	279	360	441	522	603	684	765	846
	4	400	528	720	912	1104	1296	1488	1680	1872	2064
	5	900	1150	1525	1900	2275	2650	3025	3400	3775	4150
	6	1764	2196	2844	3492	4140	4788	5436	6084	6732	7380
	7	3136	3822	4851	5880	6909	7938	8967	9996	11025	12054
	8	5184	6208	7744	9280	10816	12352	13888	15424	16960	18496
	9	8100	9558	11745	13932	16119	18306	20493	22680	24867	27054
	10	12100	14100	17100	20100	23100	26100	29100	32100	35100	38100

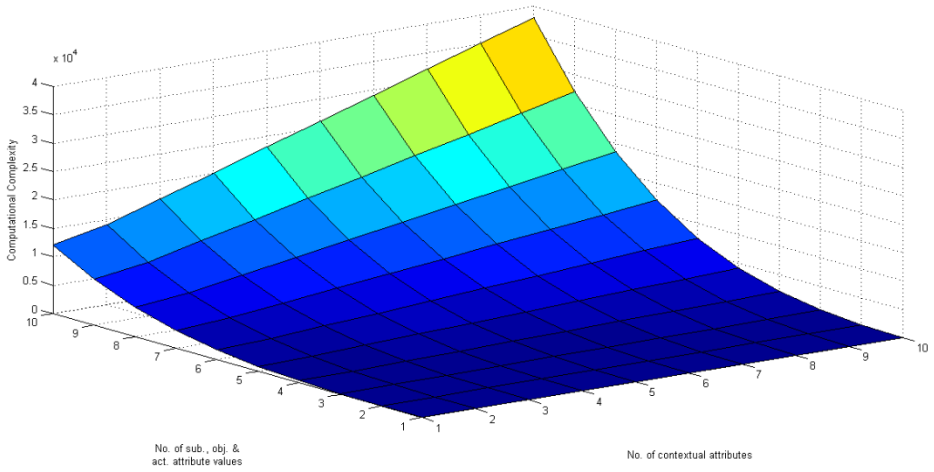


Figure 5.1. Complexity analysis of proposed algorithm for Case 1.

Table 5.2. Complexity analysis with three decision attribute values

Number of Subject, Object and Action attribute values	Number of Contextual Attributes									
	1	2	3	4	5	6	7	8	9	10
1	12	18	27	36	45	54	63	72	81	90
2	108	156	228	300	372	444	516	588	660	732
3	432	594	837	1080	1323	1566	1809	2052	2295	2538
4	1200	1584	2160	2736	3312	3888	4464	5040	5616	6192
5	2700	3450	4575	5700	6825	7950	9075	10200	11325	12450
6	5292	6588	8532	10476	12420	14364	16308	18252	20196	22140
7	9408	11466	14553	17640	20727	23814	26901	29988	33075	36162
8	15552	18624	23232	27840	32448	37056	41664	46272	50880	55488
9	24300	28674	35235	41796	48357	54918	61479	68040	74601	81162
10	36300	42300	51300	60300	69300	78300	87300	96300	105300	114300

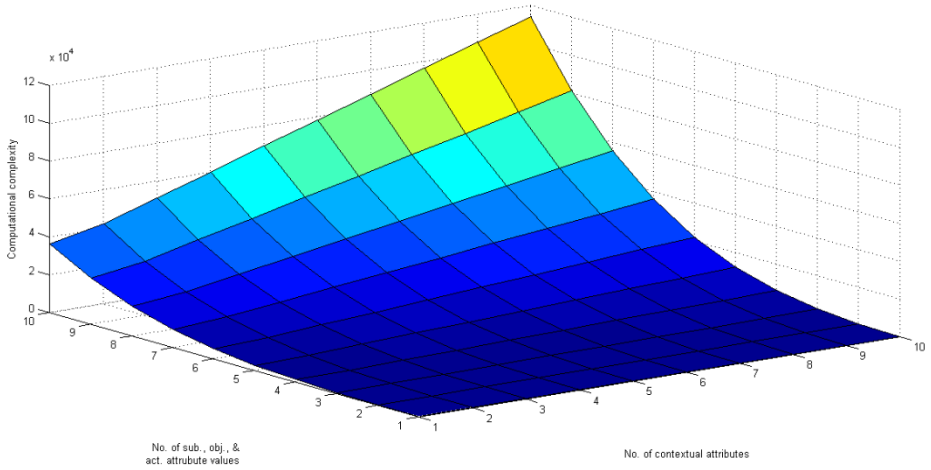


Figure 5.2. Complexity analysis of proposed algorithm for Case 2.

5.2. Qualitative Comparison

We have compared the proposed solutions on the basis of their effectiveness and the method adopted for the verification and validation of ACPs. Following are the main attributes considered for the comparison of the proposed solutions.

- **Inconsistency**

This attribute defines whether the method proposed by the authors for the validation of ACPs detect the inconsistency problems in them or not. We also consider that whether it only provides a mechanism for detection of inconsistencies or provides the solution to remove these inconsistencies.

- **Approach**

Under this heading we have defined the approach used by the authors to validate the policies. We have classified the solutions proposed by different authors on the basis of the approaches adopted by them for validation purpose.

- **Boolean Expression**

This attribute deals with the rules defined in the policies. It is used to check whether the proposed solution is applicable to simple rules or it involves some conditional attributes as well.

- **Discrete/Continuous Attribute Values**

It is clear from the attribute name that whether the proposed solution deal with the discrete data or it considers the continuous case as well. In some cases, the data of both these kinds are considered for validation.

- **Handling of Static/Dynamic Data**

In some cases, the rules defined in policies do not change at run time but in some cases these may change. So it is very important to check whether the proposed solution is applicable to both the scenarios or it may deal with any one of them.

- **Contextual Attributes**

Some attributes defined in the rules state that those rules are applicable in specific contexts. For example time, date etc., which states that an access may be granted on some resources for a specific time period.

In Table 5.3, we have summarized the work done by different researchers for the validation of access control policies. We have compared their work with respect to its efficiency and effectiveness in validation of access control policies. We can see that most of the researchers have worked on the inconsistency problem whether it is related to the detection or resolution or both. Only few of them have addressed the

incompleteness problem and it is also limited to the detection of incompleteness problem. Ma et al. [10], R. Abbasi and S.G.E Fatmi [15] have proposed the methods which are capable of detection of both, inconsistency and incompleteness, whereas Shaikh *et al.* in [3], [61] have proposed a method to detect inconsistencies which is capable of handling Boolean expressions and contextual attributes. Furthermore it is applicable to the dynamic data as well. Similarly in [4] they have proposed a method for detection of incompleteness. Stepien *et al.* [12] and Sun *et al.* [13] also have proposed methods to deal with the inconsistency and both of these are capable of handling Boolean expressions and contextual attributes.

Table 5.3. Comparison of different approaches to validate the ACPs

Author(s)	Inconsistency	Boolean Expression	Approach	Continuous / Discrete	Static / Dynamic	Contextual Attributes
Our Proposed Method	Detection + Resolution	Yes	Decision Tree based Algorithm	Both	Dynamic	Yes
Wang *et al.* [2]	Prevention	No	Formal method	Discrete	Static	No
Shaikh *et al.* [3]	Detection	Yes	Data classification	Both	Both	Yes
Shaikh *et al.* [4]	No	Yes	Data classification	Both	Static	Yes
Hwang *et al.* [5]	No	Yes	Symbolic model checker NuSMV	Both	Both	Yes
E. Martin [6]	General Fault testing	No	Fault model mutation testing using Alloy	Discrete	Static	No
M. Mankai and L. Logrippo [7]	Detection	No	Model checking Alloy	Discrete	Static	No
Bei *et al.* [8]	Detection	Yes	Matrix based algorithm	Both	Static	Yes
V.R. Karimi and D.D. Cowan [9]	Detection + Resolution	No	Model checking Alloy	Discrete	Static	No

Ma *et al.* [10]	Detection	No	Model Checking SPIN	Discrete	Static	No
Mohan *et al.* [11]	Detection + Resolution	No	Foramal method	Discrete	Both	No
Stepien *et al.* [12]	Resolution	Yes	Prolog	Both	Static	Yes
Sun *et al.* [13]	Detection + Resolution	Yes	Purpose based access control model	Discrete	Static	Yes
Huang *et al.* [14]	Detection	No	Tool SAVIS, algorithm	Discrete	Static	No
R.Abbasi and S.G.E Fatmi [15]	Detection	No	Promela specification language, RG	Discrete	Static	No
Bravo *et al.* [16]	Detection + Resolution	No	DTD graph, algorithms	Discrete	Static	No
Jin-hua *et al.* [17]	Detection	Yes	IETF policy framework	Discrete	Static	No
Xu *et al.* [18]	No	Yes	Model based, Predicate / Transition (PrT) net	Discrete	Static	Yes
E. Martin and T. Xie [19]	General Fault Testing	No	Fault Model Mutation testing	NA	Static	No
Shafiq *et al.* [20]	Detection + Resolution	No	Integer Programming technique, graphs, algorithm	Yes	Static	No
Tekbacak *et al.* [21]	General Fault Testing	No	XACML framework for ACPs	NA	Static	No
Fisler *et al.* [30]	Detection + Resolution	Yes	Decision diagrams MTBDD	Discrete	Static	No
Mukamala *et al.* [51]	Detection	No	Role-mining approach	Discrete	Static	No
Bauer *et al.* [53]	Detection + resolution	Yes	Association rule mining approach	Discrete	Both	No
Evan Martin and Tao Xie [55]	Detection	No	Data Mining Approach	Discrete	Static	No
Rémi Delmas and Thomas Polacsek [58]	Detection	No	Logical Modelling Framework	Discrete	Static	No

All the details about the validation methods have shown in the Table 5.3, which help us to compare them on the basis of the attributes used for comparison purpose. Results obtained from this comparison are helpful for the readers to decide what kind of techniques could be used to solve different type of problems. Furthermore it also helps us to choose the most appropriate technique for this purpose. It also gives us an idea about the issues in access control policies addressed by different researchers. For example most of the researchers have focused on detection and/or resolution of inconsistency problems in access control policies but each of them have some limitations. Only few of them have addressed all the issues. It is also clear from the results that the less focus is given on the issue of handling of contextual attributes.

Chapter 6

Conclusion and Future Work

6.1. Discussion

We have discussed different access control policy verification and validation frameworks proposed by different researchers by using different approaches. We also classified the work done by others according to the approaches used for validation purpose. We have categorized these methods based on the proposed taxonomy. We also have compared existing methods on the basis of different attributes that gives a clear view about those approaches and their ability to deal with different kind of issues in the access control policies. The comparison of different techniques shows that most of these policy validation schemes have focused on inconsistency detection. Some tools have also been developed to implement the techniques proposed by researchers to provide the mechanisms to resolve the issues related to the policy validation. Although some techniques are very efficient and helpful to resolve these issues but still a lot of work is required because most of them do not handle the policies that

contains Boolean expressions and contextual discrete or continuous attributes. Keeping in mind all these facts we have proposed a method to detect inconsistencies in access control policies that not only deals with the continuous attribute values but also provides a mechanism to deal with the issues related to the rules involving Boolean expression. In addition, it also provides a mechanism to deal with the contextual attribute values used by the administrators in the defined rules. By using graphs, we have tried to elaborate those issues which are required to be given more attention. Different trend graphs also show the work done by researchers in this area using various approaches. A lot of work has done in this area but still there are many issues left that need researcher's attention.

6.2. Conclusion

We have proposed an algorithm to detect and resolve the inconsistencies in the access control policies and have used the decision tree approach for validation purpose. We also have developed a software to validate the access control policies by implementing the proposed algorithm. It provides a solution to validate the access control policies especially those which involve contextual attributes and expressions that involve the comparison operators. By supporting Boolean expressions, continuous attribute values and contextual attribute values, our proposed algorithm also reduces the number of rules. But this approach also has some limitations. For example, this algorithm supports bounded continuous attribute values and does not provide any solution for detection and resolution of incompleteness problem. We also have given the complexity analysis of our proposed method which shows the exponential growth of the complexity curve. Increase in complexity is dependent on the number of distinct

decision attribute values. So we need to address these issues and also to improve the performance in terms of computational complexity.

6.3. Future Work

In this work we have proposed an algorithm to detect and resolve inconsistencies in the access control policies. Our proposed algorithm takes the rules defined by the administrators as an input in the form of a decision tree. In our developed system currently, it only takes the rules defined in the XML. So in future, first we will provide the support to accept the rules as input defined in other formats like text file as well. Our current algorithm can only detect and resolve the inconsistencies in the access control policies but in future we plan to propose another algorithm to detect and resolve the incompleteness issue. This may done as part of this algorithm or may be in the form of a new algorithm. In this algorithm we have provided support to handle continuous values to some degree (bounded continuity), but in future we would like to handle more complex forms. In addition, we would like to reduce the complexity of the proposed algorithm to improve its efficiency.

LIST OF REFERENCES

1. Pierangela Samarati, Sabrina De Capitani di Vimercati, "Access Control: Policies, Models and Mechanisms", R. Focardi and R. Gorrieri (Eds.): FOSAD 2000, LNCS 2171, pp. 137–196, 2001.
2. Yigong Wang, Hongqi Zhang, Xiangdong Dai, Jiang Liu, "Conflicts Analysis and Resolution for Access Control Policies", *IEEE Int. Conf. on Information Theory and Information Security (ICITIS),* 2010, pp. 264-267.
3. Riaz Ahmed Shaikh, Kamel Adi, Luigi Logrippo, Serge Mankovski, "Inconsistency Detection Method for Access Control Policies", in Proc. of *Sixth International Conference on Information Assurance and Security*, 2010, pp. 204-209.
4. Riaz Ahmed Shaikh, Kamel Adi, Luigi Logrippo, Serge Mankovski, "Detecting Incompleteness in Access Control Policies using Data Classification Schemes", *Fifth Int. Conf. on Digital Information Management (ICDIM),* 2010, pp. 417-422.
5. YeeHyun Hwang, Tao Xie, Vincent Hu, Mine Altunay, "ACPT: A Tool for Modeling and Verifying Access Control Policies", *IEEE International Symposium on Policies for Distributed Systems and Networks*, 2010, pp. 40-43.
6. Evan Martin, "Testing and Analysis of Access Control Policies", in Proc. of 29th International Conference on Software Engineering, 2007, pp. 75-76.
7. Mahdi Mankai, Luigi Logrippo, "Access Control Policies: Modeling and Validation", in Proc. of the *5th NOTERE Conference*, Canada, August 2005, pp. 85-91.

8. WU Bei, CHEN Xing-yuan, ZHANG Yong-fui, DAI Xiang-dong, "An Extensible Intra Access Control Policy Conflict Detection Algorithm", *International Conference on Computational Intelligence and Security*, 2009, pp. 483-488.
9. Vahid R. Karimi, Donald D. Cowan, "Verification of Access Control Policies for REA Business Processes", *33rd Annual IEEE International Computer Software and Application Conference*, 2009, pp. 422-427.
10. Jianli Ma, Dongfang Zhang, Guoai Xu, Yixian Yang, "Model Checking Based Security Policy Verification and Validation", *2nd International Workshop on Intelligent Systems and Applications (ISA)* , 2010, pp. 1-4.
11. Apurva Mohan, Douglas M. Blough, Tahsin Kurc, Andrew Post, Joel Saltz, "Detection of Conflicts and Inconsistencies in Taxonomy-based Authorization Policies", *IEEE International Conference on Bioinformatics and Biomedicine*, 2011, pp. 590-594.
12. Bernard Stepien, Stan Matwin, Amy Felty, "Strategies for Reducing Risks of Inconsistencies in Access Control Policies", *International Conference on Availability, Reliability and Security, IEEE,* 2010, pp. 140-147.
13. Lili Sun, Hua Wang, Xiaohui Tao, Yanchun Zhang, Jing Yang, "Privacy Preserving Access Control Policy and Algorithms for Conflicting Problems", *International Joint Conference of IEEE TrustCom, 2011*, pp. 250-257.
14. Chao Huang, Jianling Sun, Xinyu Wang, Yuanjie Si, "Inconsistency Management of Role Base Access Control Policy", *International Conference on E-Business and Information System Security,* 2009, pp. 1-5.
15. Ryma Abassi, Sihem Guemara El Fatmi, "An Automated Validation Method for Security Policies: the firewall case", *The Fourth International Conference on Information Assurance and Security,* 2008, pp. 291-294.

16. Loreto Bravo, James Cheney, Irini Fundulaki, "ACCOn: Checking Consistency of XML Write-Access Control Policies", In proceedings of the *11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT, 2008, pp. 715-719.
17. WU Jin-hua, CHEN Xiao-su, ZHAO Yi-zhu, NI Jun, "A Flexible Policy-Based Firewall Management Framework", *International Conference on Cyberworlds*, 2008, pp. 192-194.
18. Dianxiang Xu, Lijo Thomas, Michael Kent, Tejeddine Mouelhi, Yves Le Traon, "A Model-Based Approach to Automated Testing of Access Control Policies" *SACMAT*, 2012, pp. 209-218.
19. Evan Martin, Tao Xie, "A Fault Model and Mutation Testing of Access Control Policies", *International world Wide Web Conference Committee*, 2007, pp. 667-676.
20. Basit Shafiq, Jaideep Vaidya, Arif Ghafoor, Elisa Bertino, "A Framework for Verification and Optimal Reconfiguration of Event-driven Role Based Access Control Policies", *SACMAT*, 2012, pp. 197-208.
21. Fatih Tekbacak, Tugkan Tuglular, Oguz Kikenelli, "An Architecture for Verification of Access Control Policies with Multi Agent System Ontologies", *33rd Annual IEEE International Computer Software and Application Conference*, 2009, pp. 52-55.
22. S. Murphy, G. Weber, M. Mendis, H. Chueh, S. Churchill, J. Glaser and I. Kohane, "Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2)," *Journal of the American Medical Informatics Association, 17(2)*, 2010, pp. 124-130.

23. Genrich, H.J. “Predicate/transition nets. In Petri Nets: Central Models and Their Properties”, Springer Berlin Heidelberg, 1987, pp. 207–247.
24. Xu, D. and Nygard, K.E. “Threat-driven modeling and verification of secure software using aspect-oriented Petri nets”, *IEEE Trans. On Software Engineering*, 2006, vol. 32, no. 4, pp 265-278.
25. Xu, D. “A tool for automated test code generation from high-level Petri nets”, In Proc. of *Petri Nets’11, LNCS 6709*, Newcastle upon Tyne, UK, June 2011, pp. 308-317.
26. Xu, D., Tu, M., Sanford, M., Thomas, L., Woodraska, D., and Xu, W. “Automated security test generation with formal threat models” *IEEE Trans. on Dependable and Secure Computing*. In press, *9*(4), pp. 526-540, 2012.
27. J. R. Quinlan, “Induction of decision trees,” *Mach. Learn*, vol. 1, no. 1, pp. 81–106, March 1986.
28. ——, “C4.5: Programs for Machine Learning”. USA: Morgan Kaufmann Publishers, 1993.
29. B. Cestnik, I. Kononenko, and I. Bratko, “Assistant 86: A knowledge elicitation tool for sophistical users,” in Proc. of the *2nd European Working Session on Learning*, 1987, pp. 31–45.
30. K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, “Verification and change-impact analysis of access-control policies,” in Proc. of the *27th Int. conference on Software engineering*, NY, USA, 2005, pp. 196–205.
31. M. G. Gouda and A. X. Liu, “Structured firewall design,” *Computer Networks*, vol. 51, no. 4, pp. 1106–1120, 2007.

32. J. Catlett, “Megainduction: Machine learning on very large databases,” PhD Thesis, School of Computer Science, University of Technology, Sydney, Australia, 1991.
33. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. “NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking”. In Proc. of *14th International Conference on Computer Aided Verification (CAV)*, 2002, pp. 359-364.
34. D. Jackson, “ALLOY Home Page.” [Online]. Available: http://alloy.mit.edu/
35. ——, Micromodels of Software: Lightweight Modelling and Analysis with ALLOY, Feb. 2002.
36. ——, ALLOY 3.0 Reference Manual, May 2004.
37. D. Jackson, I. Schechter, and H. Shlyahter, “Alcoa: the alloy constraint analyzer”, In proceedings of the *22nd international conference on Software engineering*. ACM Press, 2000, pp. 730–733.
38. Robert Tarjan, “Depth-first search and linear graph algorithms”, *In SIAM Journal on Computing*, Vol. 1 (1972), No. 2, pp. 146-160.
39. B.W. Lampson. “Protection”, In *5th Princeton Symposium on Information Science and Systems*, 1971, pp. 437–443.
40. G.S. Graham and P.J. Denning, “Protection principles and practice”, In *AFIPS Press, editor, Proc. Spring Jt. Computer Conference*, volume 40, Montvale, N.J., 1972, pp. 417–429.
41. M.H. Harrison, W.L. Ruzzo, and J.D. Ullman, “Protection in operating systems”, *Communications of the ACM*, 1976, pp. 461–471.
42. D.E. Denning. “A lattice model of secure information flow”, *Communications of the ACM*, Vol. 19, No. 5, May 1976, pp. 236–243.

43. D.E. Bell and L.J. LaPadula, "Secure computer systems: Mathematical foundations", *Technical Report ESD-TR-278*, vol. 1, The Mitre Corp., Bedford, MA, 1973.
44. G. Ahn and R. Sandhu, "The RSL99 language for role-based separation of duty constraints", In Proc. of the *fourth ACM Workshop on Role-based Access Control*, Fairfax, VA, USA, October 1999, pp. 43–54.
45. T. Jaeger and A. Prakash, "Requirements of role-based access control for collaborative systems", In Proc. of the *first ACM Workshop on Role-Based Access Control*, Gaithersburg, MD, USA, November 1995.
46. G. Lawrence, "The role of roles", *Computers and Security*, Vol. 12, No. 1, 1993, pp. 15-21.
47. C. Powers and M. Schunter, "Enterprise privacy authorization language (EPAL 1.2)", W3C Member Submission, November 2003.
48. F. Somenzi, "CUDD: The CU decision diagram package", http://vlsi.colorado.edu/~fabio/CUDD/.
49. K.J. Biba, "Integrity considerations for secure computer systems", Technical Report TR-3153, The Mitre Corporation, Bedford, MA, April 1977.
50. T. Moses, "eXtensible Access Control Markup Language (XACML) version 1.0", Technical report, OASIS, Feb. 2003.
51. Ravi Mukkamala, Vishnu Kamisetty, Pawankumar Yedugani, "Detecting and Resolving Misconfigurations in Role-Based Access Control", *ICISS 2009*, pp. 318-325.
52. Vaidya, J., Atluri, V., Guo, Q., "The Role-Mining Problem: Finding a Minimal Descriptive Set of Roles", In proc. of *12 ACM Symposium on Access Control Models and Technologies*, ACM Press, New York, 2007, pp. 175–184.

53. Lujo Bauer, Scott Garriss, Michael K. Reiter, “Detecting and Resolving Policy Misconfigurations in Access-Control Systems”, *ACM Transactions on Information and System Security (TISSEC) 14.1 (2011): 2*.
54. R. Agrawal and R. Srikant. “Fast algorithms for mining association rules”, In Proceedings *20th International Conference on Very Large Data Bases, VLDB*, 1994, pp. 487-49.
55. Evan Martin and Tao Xie, “Inferring Access-Control Policy Properties via Machine Learning”, proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks, 2006.
56. I. H. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 2005.
57. Sun Microsystems. Sun’s XACML Implementation. Source-forge, 2005.
58. Remi Delmas and Thomas Polacsek, “Formal Methods for Exchange Policy Specification”, CAiSE, 2013, pp. 288-303.
59. Gallier, J.H.,”Logic for Computer Science: Foundations of Automatic Theorem Proving”, ch. 10, pp. 448–476, Wiley, 1987.
60. E. Rissanen, “eXtensible Access Control Markup Language (XACML) Version 3.0 OASIS Standard.” http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf, Jan 2013. Accessed: 2014-02-03.
61. Riaz Ahmed Shaikh, Kamel Adi, Luigi Logrippo, Serguei Mankovski, "Validation of Consistency and Completeness of Access Control Policy Sets" Us 2012/0124639 A1, May 17, 2012.
62. Eric Yuan, Jin Tong, “Attributed Based Access Control (ABAC) for Web Services”, In proceedings of the IEEE International Conference on Web Services, ICWS, 2005.

63. Giovanni Russello, Changyu Dong, Naranker Dulay, “A Workflow-based Access Control Framework for e-Health Application”, 22nd International Conference on Advanced Information Networking and Applications, IEEE, 2008.
64. David F.C. Brewer, Michael J. Nash “The Chinese Wall Security Policy”, IEEE, 1989, pp. 206-214.