I.J. Computer Network and Information Security, **2015, 1, 54-69**
Published Online December 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2015.01.08

Analysis and Comparison of Access Control Policies Validation Mechanisms

Muhammad Aqib, Riaz Ahmed Shaikh
Computer Science Department, King Abdulaziz University, Jeddah, 21589, Saudi Arabia
Email: aqib.qazi@yahoo.com, rashaikh@kau.edu.sa

Abstract—Validation and verification of security policies is a critical and important task to ensure that access control policies are error free. The two most common problems present in access control policies are: inconsistencies and incompleteness. In order to detect such problems, various access control policy validation mechanisms are proposed by the researchers. However, comprehensive analysis and evaluation of the existing access control policy validation techniques is missing in the literature. In this paper, we have provided a first detailed survey of this domain and presented the taxonomy of the access control policy validation mechanisms. Furthermore, we have provided a qualitative comparison and trend analysis of the existing schemes. From this survey, we found that only few validation mechanisms exist that can handle both inconsistency and incompleteness problem. Also, most of the policy validation techniques are inefficient in handling continuous values and Boolean expressions.

Index Terms—Access control, Inconsistency, Incompleteness, Policy Validation, Policy Verification.

I. Introduction

Security of enterprise applications is a critical issue, and the lapse in their security may disclose the confidential data and information to the unauthorized users. To protect the data and resources from unauthorized access, monitoring and controlling mechanism should be enforced. For this purpose, different kinds of access control policies (ACPs) are implemented that are broadly categorized into three types: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-based Access Control (RBAC) [1]. The main purpose of these policies is to protect the data and resources of the system from the unauthorized access. These policies basically define that which user has the access to which resources under which specific conditions, e.g. time, day. These policies are mostly defined by the policy administrators.

Mainly two kinds of problems exist in access control policies that are inconsistency and incompleteness problems. Inconsistency problem in ACPs arise when two or more policies defined by the administrators lead to the contradictory outputs. For example, one policy may allow a user to access a certain resource under certain conditions while the other policy may restrict the same user from accessing the same resource under the same conditions. Incompleteness problem arises when some situations in the system exists for which no policy rule is defined by the system administrators. Many policy validation approaches [4], [7], [14], [30] have been proposed by different researchers that deal with the problems of inconsistency and incompleteness in ACPs. However, comprehensive analysis and evaluation of the existing access control policy validation techniques is missing in the literature.

In this paper, we have summarized different approaches adopted by the different researchers to detect and resolve inconsistencies and incompleteness issues in ACPs. We have examined existing approaches on the basis of different attributes to test the effectiveness of the proposed solutions. Our contribution is fourfold:

1. We have provided comprehensive survey on access control policy validation techniques. To the best of our knowledge this is the first comprehensive survey paper on policy validation techniques.
2. We have presented taxonomy for access control policy validation techniques.
3. We have provided qualitative comparison of the existing policy validation techniques.
4. We have also provided trend analysis, which identifies most common and new emerging techniques used for the policy validation.

The rest of the paper is organized as follows. Section 2 contains a brief description of the access control policies and their problems. Section 3 presented the proposed taxonomy of access control validation techniques. Section 4 provides the qualitative comparison and trend analysis of the existing ACP validation techniques and finally, section 5 concludes the paper.

II. 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, a student can access the system to view his attendance, marks and grades, courses available for registration etc., but she is not allowed to mark her attendance, change her 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.

A. Types of Access Control Policies

As mentioned earlier, there are three different types of access control models and each one has its own characteristics. These three [1] types of models are: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access Control (RBAC). In this section we will briefly describe these models.

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].

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] [64]. Bell-Lapadula model [43] (for confidentiality) and Biba model [49] (for integrity) are the two common examples of MLS models.

The 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].

B. Problems with Access Control Policies

As discussed earlier, ACPs play an important role in the system to ensure the secrecy and integrity of the data. Error-prone ACPs make the system resources vulnerable to unauthorized access. Mainly two kinds of problems are discussed in policy validation mechanisms that are inconsistency and incompleteness problems.

Inconsistency: Let *S, O* and *A* are the sets of subjects, objects and actions respectively. Let $a \in A$ be the action performed by the subject $s \in S$ on the object $o \in O$. Let $d \in D$ is the decision taken on the basis of information provided by the rule *r* where $D = \{permitted, denied, undefined\}$ and a rule $r \in R$ is a three tuple rule $(s, o, a) \rightarrow d$. A policy is said to be inconsistent if for any two rules r_i and $r_j \in R$ such that $i \neq j$, r_i and r_j lead to contradictory decisions, i.e $r_i \rightarrow d_i$, $r_j \rightarrow d_j$ and $d_i \neq d_j$.

Table 1. Access control rules for different roles

Rule	Subject	Object	Action	Decision
1	Manager	File 1	Read	Permitted
2	Manager	File 1	Write	Permitted
3	Manager	File 1	Delete	Permitted
4	Clerk	File 1	Read	Permitted
5	Clerk	File 1	Write	Permitted
6	Clerk	File 1	Delete	Denied

Table 2. New rules describing new rights assigned to clerk.

Rule	Subject	Object	Action	Decision
7	Clerk	File 1	Read	Permitted
8	Clerk	File 1	Write	Permitted
9	Clerk	File 1	Delete	Permitted

Table 3. No rule defined to cancel registration on Saturday.

Rule	Subject	Object	Action	Decision	Day
1	Manager	File 1	Register new	Permitted	Mon , Tue, Wed, Thu, Fri
2	Manager	File 1	View status	Permitted	
3	Manager	File 1	Cancel registration	Permitted	
4	Clerk	File 1	Register new	Permitted	
5	Clerk	File 1	View status	Permitted	
6	Clerk	File 1	Cancel registration	Denied	
7	Clerk	File 1	View status, Register new	Permitted	Sat

Example of Inconsistency: Let us consider the example of a clerk and his manager working in a company. The manager has the rights to read, update and delete the contents of a resource file. The clerk can read and update the contents of that file but he has no rights to delete its contents as shown in the Table 1. It is clear from rule 6 that the clerk has no right to perform delete operation on File1. Let us assume that manager has delegated his access rights to the clerk. Then the new rules defined in the Table 2 will be added to the policy set. Now according to the new rules defined in Table 2, clerk is allowed to perform the delete operation as well on File 1, which is contradictory to rule 6 in Table 1, which does not allow him to perform the delete operation. This is a very simple example of inconsistencies in the access control policies.

Incompleteness: A policy set *R* is said to be incomplete if there exists some possible rule *r* for some situation such that $r \notin R$.

Example of Incompleteness: Consider the seven rules defined in Table 3. There is a manager and a clerk in an office. Clerk has assigned the duty to register new customers and he can also check their status but he has no right to cancel the registration of any customer. Office is opened six days a week but manager comes to the office for five days. It is mentioned that the clerk can view status of an existing customer and can also register new customers on Saturday. But it has not been defined in any rule that whether he is allowed to cancel registration of a customer on Saturday or not. These problems are known as incompleteness problems and may be harmful for the security of a system.

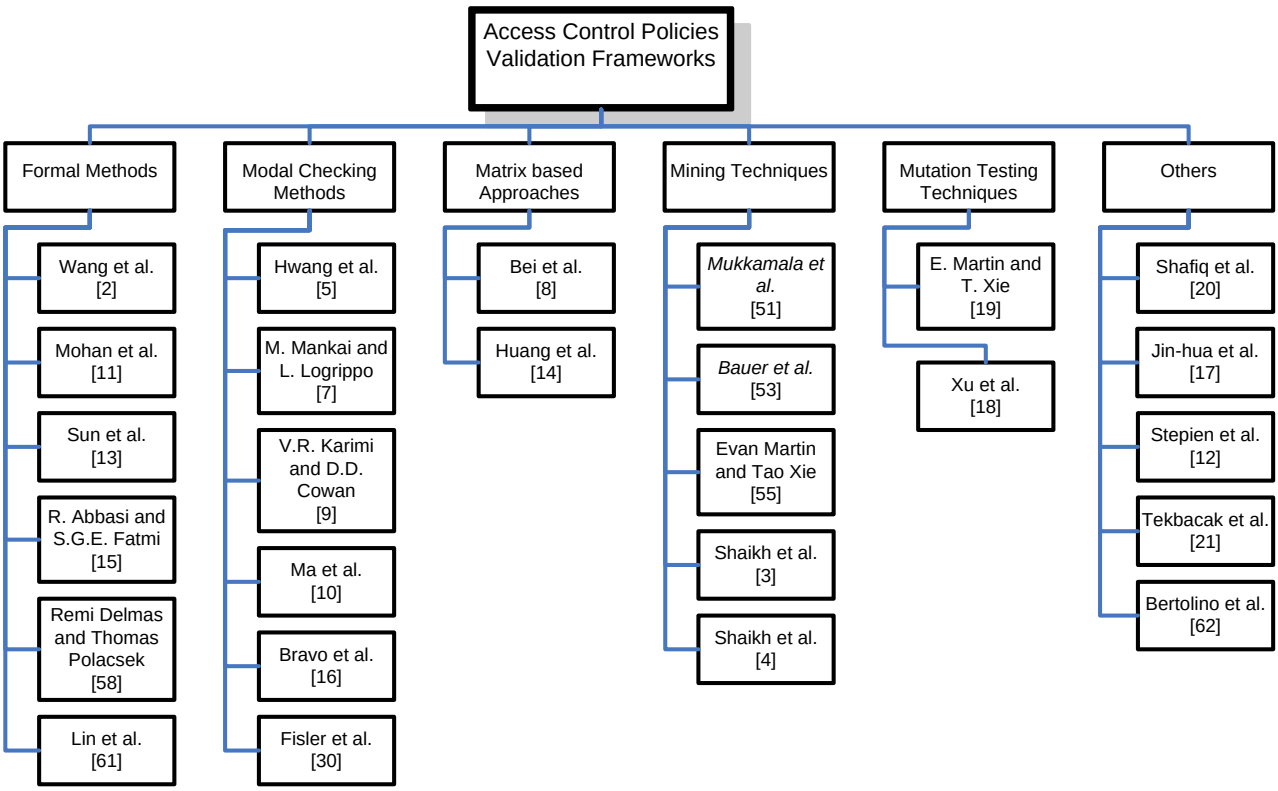

Fig. 1 Classification of different approaches for validation of ACP

III. Classification of ACPs Validation Frameworks

Different approaches have been adopted by the different authors to address the ACPs verification and validation issues. In this section, we have presented some of the methods and frameworks proposed for policy validation. As shown in Figure 1, we have classified the proposed methods into the following six categories.

A. Mining Techniques
B. Model Checking Techniques
C. Formal Methods
D. Matrix-based Approaches
E. Mutation Testing Approaches
F. Other Techniques

A. 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 terms of under-privileges and over-privileges to discuss the misconfigurations in the access control polices. Furthermore, top-down and bottom-up approaches have been discussed which are normally used to address these problems. The authors have used the bottom-up approach, also called role-mining problem. They have used a tiling approach proposed in [52] to discover roles by using privileges. 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. Authors claim 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.

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. 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.

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 sorts 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.

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 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. After creation of the decision tree, an inconsistency detection 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 show that the proposed solution can efficiently detect inconsistencies in both cases. While dealing with Boolean expressions, the proposed solution provides the solution for contextual attributes values as well.

Shaikh *et al.* in [4] have provided a mechanism to detect incompleteness in ACPs using data classification techniques. 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 modified version of C 4.5 algorithm has reduced ordered complexity as compared to the original algorithm. The proposed method is only limited to the detection of the incompleteness in ACPs. It also deals with the expressions involving Boolean variables and contextual attribute values. Furthermore, it also deals with continuous attribute values and dynamic data in policy rules.

B. 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] for policy specifications. In this section we will discuss all those mechanisms which use modal checking tools.

In [5], Hwang *et al.* have developed a tool named Access Control Policies Testing (ACPT) 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 is very helpful in generating policies based upon the policy requirements but it also suffers from various limitations. 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. Modeling structure includes the definition and mapping of attributes, values, subjects, resources, actions, requests, targets, effects, combining algorithms, policies and policy sets. 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 are also defined for the target verification purpose. If a target matches a request, the response defined in logical model is returned. The proposed 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. Although the proposed solution detects inconsistencies and provides their solution to some extant as well but it only deals with policies having discrete attribute values and static data.

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 behavior 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 behavior 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. 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. The proposed system deal with both inconsistency and incompleteness detection but it does not provide any solution for these problems. Furthermore, it only deals with discrete and static data without involving any Boolean expression and contextual attribute values.

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. 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 named as Set Cover 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.

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.

C. 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.

Lin *et al.* [60] have proposed a comprehensive tool for the analysis of access control policies, called EXAM. This tool is used to detect inconsistencies and redundancies. Also, it handles continues values. Similar to Shaikh *et al.* [3],[4] and Fisler el al. [30], authors have used a variant of Multi-terminal Binary Decision Diagrams (MTBDD). The proposed tool cannot be used to detect incompleteness in access control policy sets.

Cau *et al.* [61] have proposed a framework for policy specification, verification and enforcement. In that framework, they specified the policies in fusion logic that allows users to verify various properties of access control policies such as accessibility, dynamic separation of duty, dynamic and static conflicts. The decision procedure for fusion logic is developed with the help of Binary Decision Diagrams (BDD). The proposed tool cannot be used to detect incompleteness in access control policy sets.

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 categorized 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 Pi and Pj, Pi should be assigned higher priority according to principle of specific take precedence. According to this work, priorities will be swapped between Pi and Pj and then check ACPj, 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 analyze 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.

D. 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.

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 $O(l\,m^2)$ where $l = |U|$ and $m = |R|$.

E. 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 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: Policy Set, 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 on inconsistency and incompleteness issues in depth.

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 pair-wise 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.

F. Other Techniques

Bertolino *et al.* [62] have proposed an Access Control Testing toolchain (ACT) that can be used for designing and testing access control policies. In this tool, users can specify a graphical access control model. The tool will automatically translate this model into XACML policy. Furthermore, the ACT tool can detect inconsistencies between the model and derived policies.

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 are 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 needed 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. Also, 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, policy decision point and policy administration point. 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.

IV. Comparison and Analysis

As mentioned above, 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 proposed validation method detects the inconsistency problems or not.
- **Incompleteness:** Same like inconsistency, we compare the proposed solution on the basis of incompleteness detection and the ability to resolve the incompleteness problems.
- **Approach:** Under this heading we have defined the approach used by the authors to validate the policies.
- **Boolean expression:** It deals with the expressions 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.
- **Continuous/Discrete:** 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.
- **Static/Dynamic:** 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.

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. Fig. 2 shows a trend graph for different proposed techniques during 2005-2013, which 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.

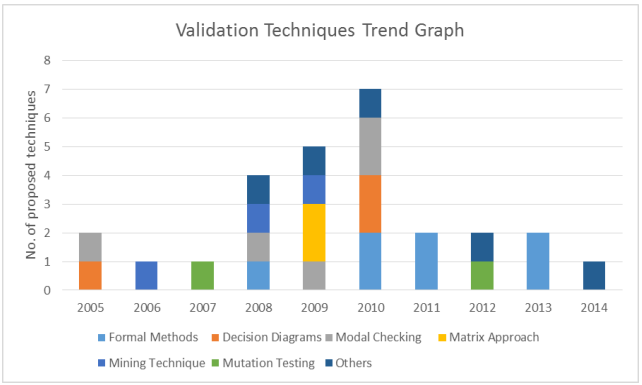

Fig. 2 Graph showing the ratio of validation methods adopted by researchers from 2005-2014

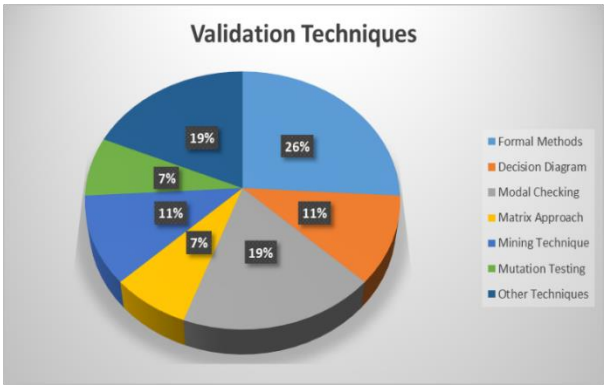

Fig. 3 The percentage distribution of different types of proposed validation techniques

Table 4. Comparison of different approaches to validate the ACPs

Papers	Inconsistency	Incomplete-ness	Boolean Expression	Approach	Continuous / Discrete	Static / Dynamic	Contextual Attributes
Wang *et al.* [2]	Prevention	No	No	Formal method	Discrete	Static	No
Shaikh *et al.* [3]	Detection	No	Yes	Data classification	Both	Both	Yes
Shaikh *et al.* [4]	No	detection	Yes	Data classification	Both	Static	Yes
Hwang *et al.* [5]	No	No	Yes	Symbolic model checker NuSMV	Both	Both	Yes
E. Martin [6]	General Fault testing	No	No	Fault model mutation testing using Alloy	Discrete	Static	No
Mankai & Logrippo [7]	Detection	No	No	Model checking Alloy	Discrete	Static	No
Bei *et al.* [8]	Detection	No	Yes	Matrix based algorithm	Both	Static	Yes
Karimi & Cowan [9]	Detection + Resolution	No	No	Model checking Alloy	Discrete	Static	No
Ma *et al.* [10]	Detection	Detection	No	Model Checking SPIN	Discrete	Static	No
Mohan *et al.* [11]	Detection + Resolution	No	No	Formal method	Discrete	Both	No
Stepien *et al.* [12]	Resolution	No	Yes	Prolog	Both	Static	Yes
Sun *et al.* [13]	Detection + Resolution	No	Yes	Purpose based access control model	Discrete	Static	Yes
Huang *et al.* [14]	Detection	No	No	Tool SAVIS, algorithm	Discrete	Static	No
Abbasi & Fatmi [15]	Detection	Detection	No	Promela specification language, RG	Discrete	Static	No
Bravo *et al.* [16]	Detection + Resolution	No	No	DTD graph, algorithms	Discrete	Static	No
Jin-hua *et al.* [17]	Detection	No	Yes	IETF policy framework	Discrete	Static	No
Xu *et al.* [18]	No	Detection + Resolution	Yes	Model based, Predicate / Transition (PrT) net	Discrete	Static	Yes
E. Martin and T. Xie [19]	General Fault Testing	No	No	Fault Model Mutation testing	NA	Static	No
Shafiq *et al.* [20]	Detection + Resolution	No	No	Integer Programming technique, graphs, algorithm	Yes	Static	No
Tekbacak *et al.* [21]	General Fault Testing	No	No	XACML framework for ACPs	NA	Static	No
Fisler *et al.* [30]	Detection + Resolution	No	Yes	Decision diagrams MTBDD	Discrete	Static	No
Mukamala *et al.* [51]	Detection	No	No	Role-mining approach	Discrete	Static	No
Bauer *et al.* [53]	Detection + resolution	No	Yes	Association rule mining approach	Discrete	Both	No
Martin & Xie [55]	Detection	No	No	Data Mining Approach	Discrete	Static	No
Delmas & Polacsek [58]	Detection	Detection	No	Logical Modelling Framework	Discrete	Static	No
Aqib & Shaikh [63]	Detection + Resolution	No	Yes	Tree based Algorithm	Both	Both	Yes

Chart presented in Fig. 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.

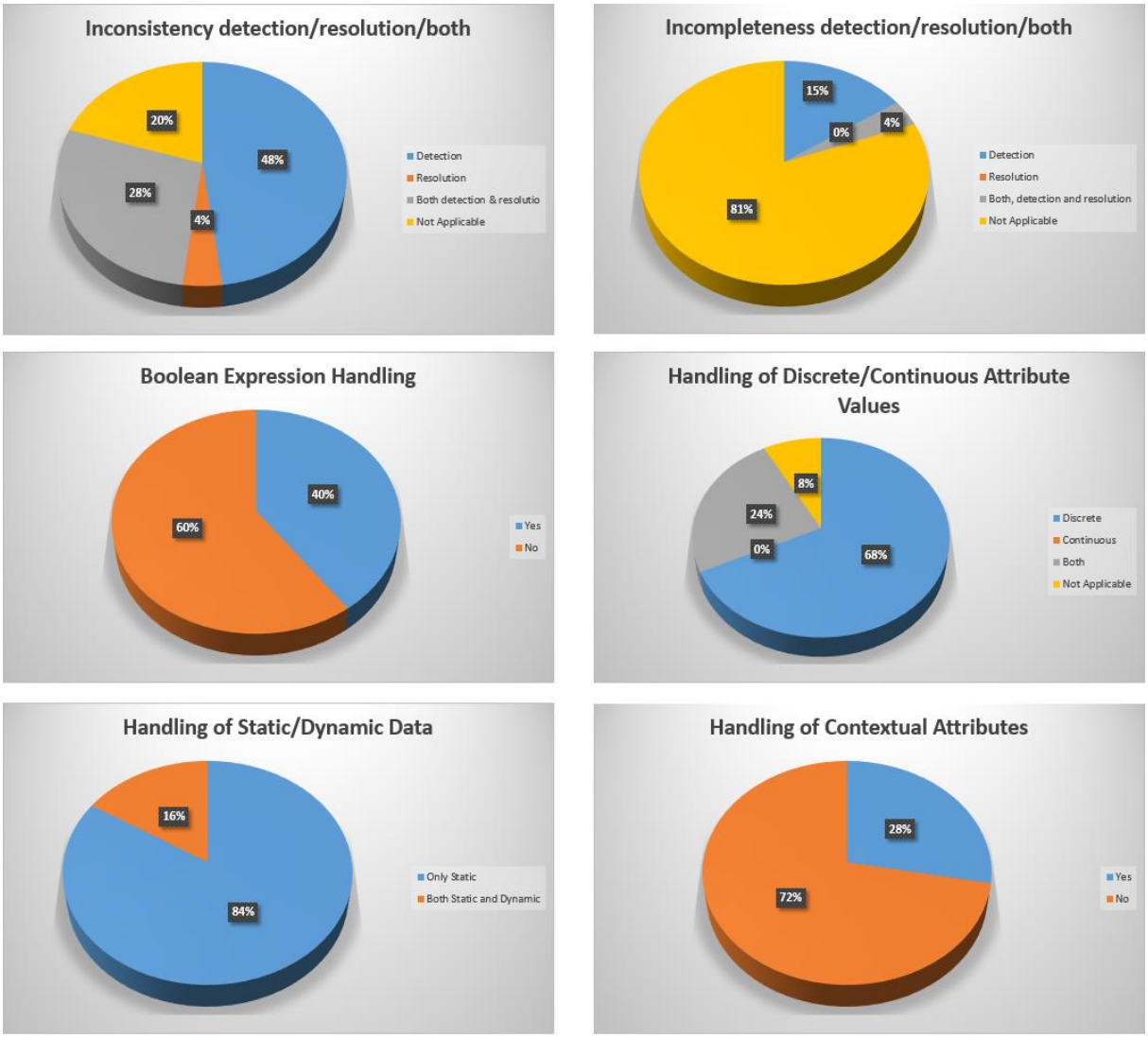

Fig 4. Percentages of issues addressed in compared techniques

In Table 4, we have summarized the work done by different researchers for the validation of ACPs. We have compared their work with respect to its efficiency and effectiveness in validation of ACPs. 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] 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.

Qualitative comparison of existing policy validation techniques is shown in Table 4. 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. Additionally, it also gives us an idea about the issues in ACPs addressed by different researchers. For example most of the researchers have focused on detection and resolution of inconsistency problems in access control policies but only few of them have addressed the incompleteness issue. It is also clear from the results that the less focus is given on the issue of handling of contextual attributes. In Fig. 4, we show a percentage distribution of the properties to show that how the researchers have addressed these issues in their proposed techniques.

V. Conclusion

In this paper, we have discussed different access control policy verification and validation frameworks proposed by different authors by using different approaches. This is the first survey paper of this domain.

We have categorized the existing methods based on the proposed taxonomy. Also, we have compared existing methods on the basis of various attributes. This comparison gives the clear view about the existing 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. Only few schemes exists which can be used to detect both inconsistency and incompleteness in access control policies. Although some techniques are very efficient and helpful to resolve these issues but still more work is needed because most of them do not handle complex policies that contains Boolean expressions and contextual discrete or continuous attributes. A lot of work has done in this area but still there are many issues left that need researcher's attention.

REFERENCES

[1] Samarati P., Vimercati S.C. de, "Access Control: Policies, Models and Mechanisms", R. Focardi and R. Gorrieri (Eds.): FOSAD 2000, LNCS 2171, pp. 137–196, 2001.

[2] Wang Y., Zhang H., Dai X., Liu J., "Conflicts Analysis and Resolution for Access Control Policies", *IEEE Int. Conf. on Information Theory and Information Security (ICITIS)*, 2010, pp. 264-267.

[3] Shaikh R.A., Adi K., Logrippo L., Mankovski S., "Inconsistency Detection Method for Access Control Policies", in Proc. of *Sixth Int. Conf. on Information Assurance and Security*, 2010, pp. 204-209.

[4] Shaikh R.A., Adi K., Logrippo L., Mankovski S., "Detecting Incompleteness in Access Control Policies using Data Classification Schemes", 5th *Int. Conf. on Digital Information Management*, 2010, pp. 417-422.

[5] Hwang J., Xie T., Hu V., Altunay M., "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] Martin E., "Testing and Analysis of Access Control Policies", in *Proc. of 29th Int. Conf. on Software Engineering*, 2007. pp. 75-76.

[7] Mankai M., Logrippo L., "Access Control Policies: Modeling and Validation", in Proc. of the *5th NOTERE Conference*, Canada, August 2005, pp. 85-91.

[8] Wu B., Chen X.n, Zjang Y., DAI Xiang-dong, "An Extensible Intra Access Control Policy Conflict Detection Algorithm",*Int. Conf. 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 Int.Computer Software and Application Conference*, 2009, pp. 422-427.

[10] Ma J., Zhang D., Xu G., Yang Y., "Model Checking Based Security Policy Verification and Validation", *2nd Int. Workshop on Intelligent Systems & Applications*, 2010, pp. 1-4.

[11] Mohan A., Blough D.M., Kurc T., Post A., Saltz J., "Detection of Conflicts and Inconsistencies in Taxonomy-based Authorization Policies", *IEEE Int. Conf. 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", *Int. Conf. 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", *Int. 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", *Int. Conf. on E-Business and Information System Security*, 2009, pp. 1-5.

[15] Abassi R., Fatmi S., "An Automated Validation Method for Security Policies: the firewall case", *The 4th Int. Conf. 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 proc. of the *11th Int. Conf. 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", *Int. Conf. on Cyberworlds*, 2008, pp. 192-194.

[18] Xu D., Thomas L., Kent M., Mouelhi T., Traon Y. L., "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", *Int. world Wide Web Conf. 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] Tekbacak F., Tuglular T., Kikenelli O., "An Architecture for Verification of Access Control Policies with Multi Agent System Ontologies", *33rd IEEE Int. Computer Software and Application Conf.*, 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] J. R. Quinlan, "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. Conf. 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 Int. Conference on Computer Aided Verification (CAV),* 2002, pp. 359-364.
[34] D. Jackson, "ALLOY Home Page." [Online]. Available: http://alloy.mit.edu/.
[35] D. Jackson, Micromodels of Software: Lightweight Modelling and Analysis with ALLOY, Feb. 2002.
[36] D. Jackson, ALLOY 3.0 Reference Manual, May 2004.
[37] D. Jackson, I. Schechter, and H. Shlyahter, "Alcoa: the alloy constraint analyzer", In proc. of the 22nd Int. Conf. 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 Conf., vol. 40, 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] Mukkamala R., Kamisetty V., Yedugani P,, "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 *12th ACM Symp. 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 Int. Conf. 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] Lin, D., Rao, P., Bertino, E., Li, N., and Lobo, J. "EXAM: a comprehensive environment for the analysis of access control policies." *International Journal of Information Security* 9(4), 2010, pp. 253-273.
[61] Cau, A., Janicke, H., & Moszkowski, B. "Verification and enforcement of access control policies". Formal Methods in System Design, 43(3), 2013, pp. 450-492.
[62] Bertolino, A., Marianne B., Said D., Francesca L., and Eda M., "A Toolchain for Designing and Testing Access Control Policies." In Engineering Secure Future Internet Services and Systems, pp. 266-286. Springer International Publishing, 2014.
[63] Aqib, M,. and Shaikh, R. A. "An Algorithm to Detect Inconsistencies in Access Control Policies ", Proc. of the Intl. Conf. on Advances In Computing, Communication and Information Technology (CCIT 2014), London, UK, June 2014, pp. 171 – 175.
[64] Hasani S. M., Modiri N., "Criteria Specifications for the Comparison and Evaluation of Access Control Models", I.J. Computer Network and Information Security IJCNIS Vol. 5, No. 5, April 2013, pp. 19-29.

Authors' Profiles

Muhammad Aqib is a student at the King Abdulaziz University, Jeddah, Saudi Arabia, and he obtained his Master degree in Computer Science from this university. He was attached to the Department of Computer Science in Faculty of Computing and Information Technology. His research interest includes privacy, security and database management.

Riaz Ahmed Shaikh is an Assistant Professor at the CS Dept. in the King Abdulaziz University, Jeddah, Saudi Arabia. He obtained his Ph.D. from Computer Engineering Dept., of Kyung Hee University, Korea, 2009, and M.S. in IT from the National University of Sciences and Technology, Pakistan, 2005. His research interest includes privacy, security, and trust management. For more information please visit http://sites.google.com/site/riaz289.