A Tool for Access Control Policy Validation

Muhammad Aqib, Riaz Ahmed Shaikh

Computer Science Department, Faculty of Computing and Information Technology,
King Abdulaziz University, Saudi Arabia
aqib.qazi@gmail.com, rashaikh@kau.edu.sa

Abstract

Inconsistency in access control policies exists when at least two rules present in the policy set lead to the contradictory decisions. It makes difficult for the system to decide which rule is applicable to the current scenario and hence make the system vulnerable to the unauthorized use. Various inconsistency detection methods have been proposed by researchers. However, those suffered from various limitations e.g., inefficient handling of numeric attributes, Boolean expressions etc. In this article, we propose a new algorithm that detects the inconsistencies in the policies using decision trees. For a proof of concept, we have developed a software tool that proves its effectiveness. Also, complexity analysis and qualitative comparison of the proposed algorithm is presented in the paper.

Keywords: Access control, Inconsistency, Policy validation, Security, XML

1 Introduction

Security of the enterprise applications is a critical issue. For this reason, different security mechanisms (such as access control) are enforced to restrict the users of the enterprise applications from an unauthorized use. In order to achieve this goal, the organization defines policies that are implemented by the administrator by defining rules in access control systems. In order to ensure that the rules do not contain any errors such as inconsistencies, enterprises adopt policy validation mechanisms.

Policy validation is not a trivial task. Many researchers [2-9] have worked on this issue and adopted various techniques to detect inconsistencies in the access policies. For example, Shaikh *et al.* [2] have used data classification techniques to detect inconsistencies in access control policies. Authors in [4-6] have used model checking technique and tools like Alloy and SPIN for this purpose. Many other approaches are also used in [7-9]. However, most of the existing solutions suffered from various limitations. For example, some solutions are only limited to discrete attributes. Some schemes are inefficient in handling continuous attributes and Boolean expressions. Some schemes are only limited to static rules etc.

In this paper, we have presented an algorithm based approach to detect inconsistencies in access control policies. Our contribution is follows:

- The proposed algorithm can efficiently handle both continuous and distribute attributes.
- The proposed algorithm can efficiently handle Boolean expression.
- The proposed algorithm can be used with static and dynamic policies.
- For the proof-of-concept we have developed "ACPs Validation Suite" that proves the effectiveness of the proposed algorithm.
- Qualitative comparison of the proposed algorithms with 12 state-of-the-art schemes is presented.

The rest of the paper is organized as follows. Section 2 defines the inconsistency and also the related concepts. Section 3 contains a description of the proposed algorithm whereas the algorithm implementation details are given in Section 4. Complexity analysis of the proposed algorithm is given in Section 5. Section 6 describes related work and qualitative comparison. Final section present conclusion and future work.

2 Concepts and Definitions

Inconsistency in the policy set exists when any two rules in that policy set lead to the contradictory outcomes. The rules defined by the administrators consist of different attribute values and the values of these attributes lead them to some decision. In the following section, we will discuss in detail about these attributes.

2.1 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. In practice, a rule must contain at least four attributes (Subject, Object, Action, and Decision). In addition to those, a policy administrator can define any finite number of

*Corresponding Author: Riaz Ahmed Shaikh; E-mail: rashaikh@kau.edu.sa
DOI: 10.3966/160792642018011901015

contextual attributes (e.g., day, time, month, age etc.). Among all these attributes, the decision attribute defines 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 $S = \{s_1, s_2, s_3, ..., s_n\}\ n \in N$, is a set of subjects containing *n* subjects, $O = \{o_1, o_2, o_3, ..., o_m\}\ m \in N$ is a set of objects containing *m* objects, $C = \{c_1, c_2, c_3, ..., c_1\}$ $l \in N$ is a set of contexts containing *l* contexts, and $A = \{a_1, a_2, a_3, ..., a_k\}\ k \in N$ is a set of actions containing *k* actions. Let $D = \{permit, deny, undefined\}$ be the set of decision attribute. 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 have same s, o, a and c attributes values and they have 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.

2.2 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 (e.g. update, delete etc.) 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 1, shows the various rules defined to perform different operations on the record file by different users. Both, Manager and Cashier can view the records in that file throughout the week, but only Manager can add new customers in the 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. Let us assume that the Manager delegates his delete record rights to the cashier. Then the rule 9 will be added in the rule set as shown in Table 2.

Now according to the new rules defined in Table 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.

Table 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 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

2.3 What is Redundancy?

When defining a rule in a policy set, it might happen that the administrator may define multiple rules to address the same scenario. As we defined in section 2.1, R is the set of rules and any two rules r_i and $r_j \in R$ such that $i \neq j$. If r_i and r_j have same s, o, a attributes values. And let C_i is the set of contextual attributes values for rule r_i and C_j is the set of contextual attributes values for rule r_j and $C_i \cap C_j \neq \phi$. Now if both the rules have same decisions i.e. $r_i \rightarrow d_x$ and $r_j \rightarrow d_y,\ \ x = y$ then these kind of rules are said to be redundant rule instead of an inconsistent rule. There

are mechanisms available to address this issue in access control policies as discussed in [26].

Algorithm proposed in next section merge all the redundant rules in one single rule. For example, we consider the rule 8 and 9 presented in Table 2. If the decision attribute value of both the rules is same i.e. either permitted or denied for both of them, then these rules are called redundant rules with overlapping contextual attribute values i.e. the value of the variable "Day". So our algorithm will consider these two rules as one single rule with the values of variable "Day" as "Mon-Fri".

3 Inconsistency Detection Algorithm

In this section, we will discuss the proposed algorithm in detail. 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 tuples, 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 phase, the 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.

3.1 Decision Tree Hierarchy

In the decision tree, the root node (at level 1) contains decision attribute nodes (at level 2) as child nodes which in turn contain action attribute nodes (at level 3) as their children. This hierarchy continues as action attribute nodes contain object attribute nodes (at level 4) and object nodes contain subject attribute nodes (at level 5) as child nodes. Finally, subject nodes contain contextual attribute nodes (at level 6) as the leaf nodes.

3.2 Inconsistencies Detection Process

As discussed above, the proposed algorithm consists of two parts that are clearly shown in Figure 1. In the following paragraphs, we will briefly describe the working of this algorithm.

Step 1. In this step, the main tree will be divided into the sub-trees equal to the number of decision attribute-values. 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 will be divided into the sub-trees equal to the number of decision attribute-values (Part A, Lines: 5-15). Suppose there are two decision attribute-values, permit and deny as shown in a sample hierarchy tree in Figure 2, 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 second 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.

Step 2. After having separate trees for each decision node as shown in the Figure 1, 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

Part - A

Input: Decision Tree

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"
```

Part - B

Function: CompareNodes

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 1. Proposed algorithm to detect inconsistencies in access control policies

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 1, 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 4.

Again as both the trees shown in Figure 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 attribute and will call the *CompareNodes* function again if the object attribute has the same values in both trees (Part B, Lines: 12-14). As shown in the Figure 4, object nodes having "File1" are same in both the trees. Now new sub-trees will be created having them as root nodes. The Figure 5 shows the resulting trees passed to the *CompareNodes* function in result of this comparison.

The *CompareNodes* function will compare the trees shown in Figure 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 6 shows the resulting sub-trees with subject attribute nodes as the root nodes.

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*

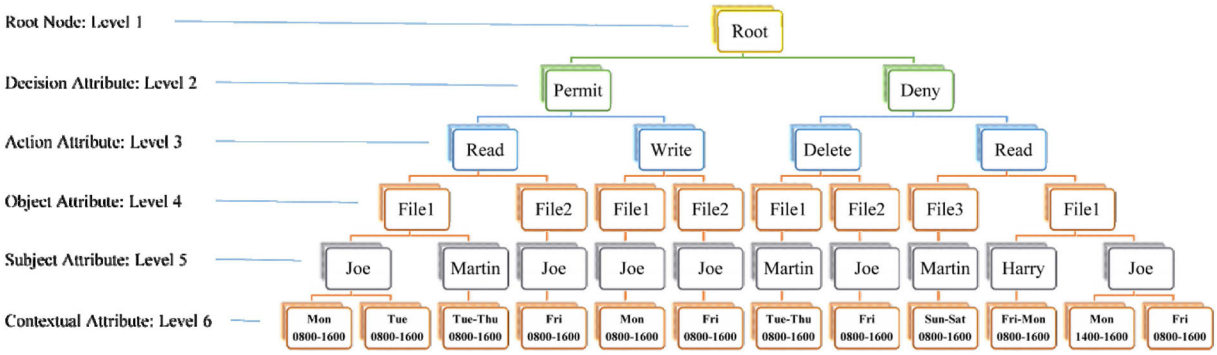

Figure 2. Sample hierarchy of the decision tree

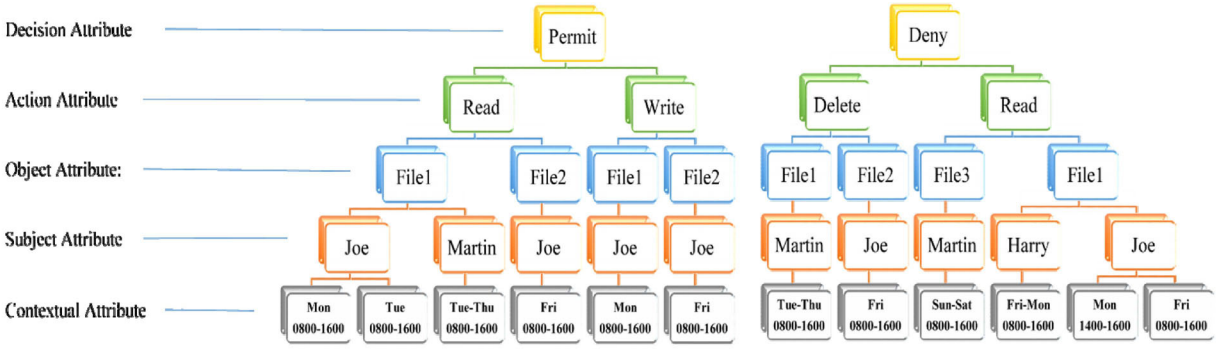

Figure 3. Sub-trees generated with decision attribute as the root node

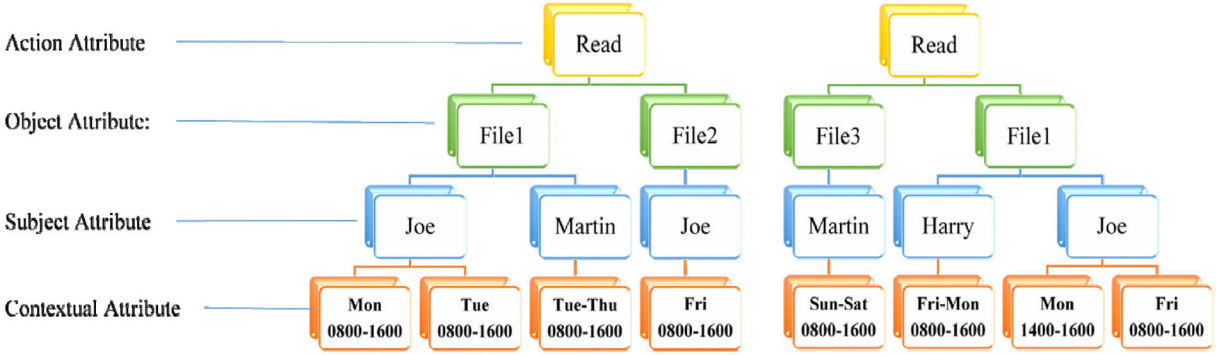

Figure 4. Sub-trees generated with action attribute as the root node

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 6, we can see that there is a contradiction in time attribute. The user "Joe" 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 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.

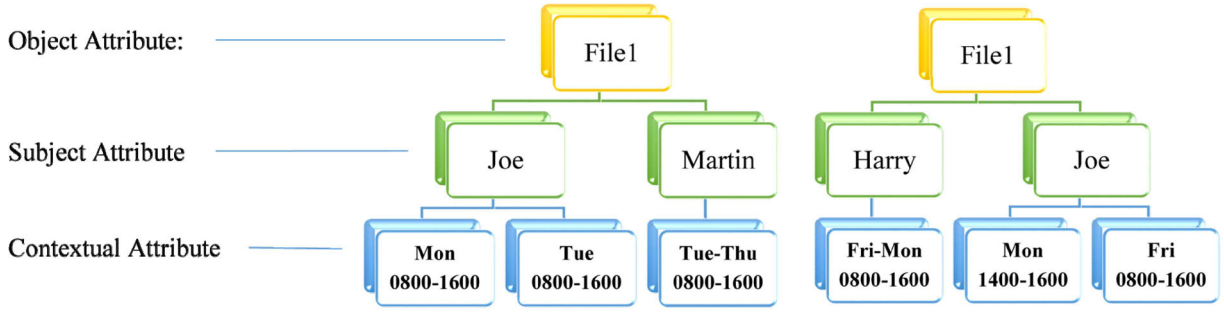

Figure 5. Sub-trees generated with object node as the root node

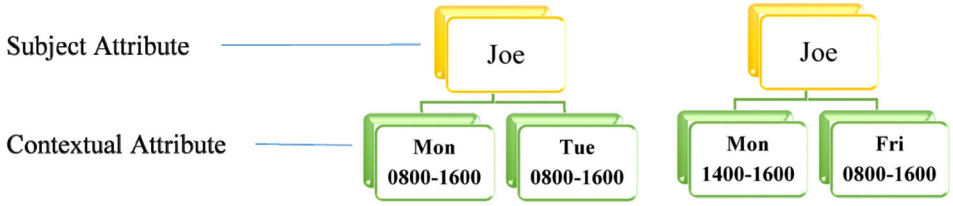

Figure 6. Sub-trees generated with subject node as the root node.

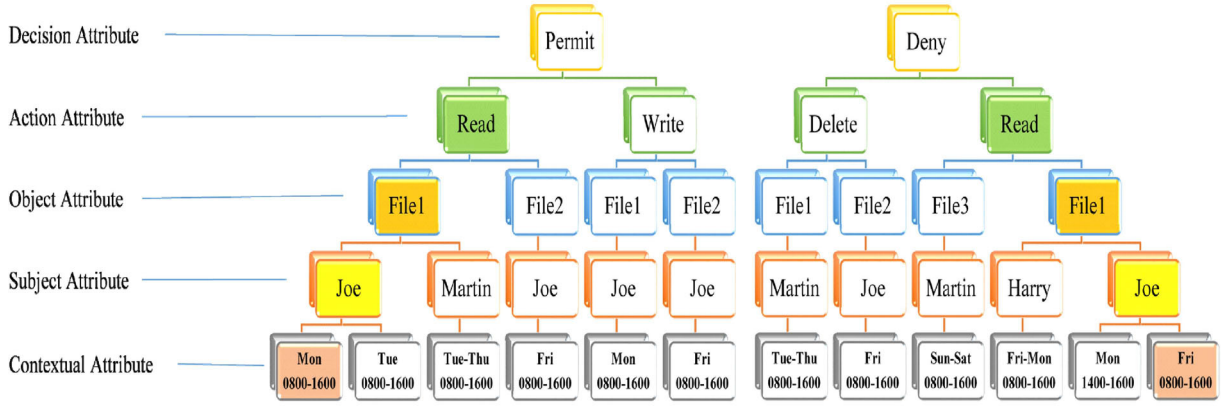

Figure 7. Rules with contradictory decisions identified

4 Algorithm Implementation

We have implemented our proposed algorithm and have developed a tool named "ACPs Validation Suite", which takes the access control policies, defined in XML file as input. By implementing the proposed algorithm, it performs the validation process and displays the inconsistent rules along with their IDs. In Figure 8, 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. In this example, we have seven distinct subject (user1 till user7), eight objects (File1 till File8), and three actions (read, write, and delete) values. Other three attributes (time, age and month) are contextual attributes. Time and age are continuous attributes,

whereas the others are discrete attributes. Figure 9 shows the ACPs Validation Suite, where the upper half of the screen shows the contents of the input XML file and the lower half shows the rules with contradictory decisions. This figure shows that tool has detected all five inconsistent rules that were present in the XML file.

```
<?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"/>
</root>
```

Figure 8. Sample XML file to detect inconsistencies in access control policies

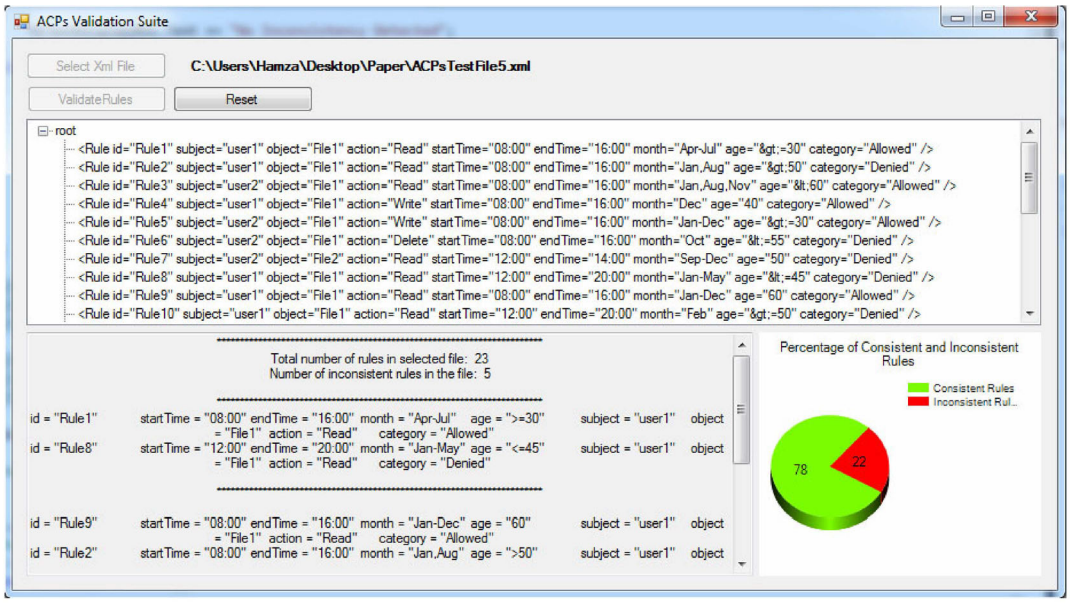

Figure 9. ACPs validation suite

4.1 Results

To evaluate the efficiency of the proposed algorithm, we have developed a simple tool that randomly generates large policy datasets in XML format. By using this tool, we have generated ten access control policy datasets. Our experiments were performed on the Intel Core i5 CPU 2.40 GHz with 6 GB RAM running on Windows 7 (64-bit operating system).

In Table 3, we have presented the details of each dataset that includes the information about number of rules and number of inconsistencies detected by ACPs Validation Suite. Figure 10 and Figure 11 show the time consumption in milliseconds and total space consumption in bytes by running the tool for each policy set respectively.

Table 3. Time and space complexity analysis

Policy Set	Number of Rules	Number of Inconsistencies	Computation Time (in ms)	Memory Usage (in bytes)
1	531	62	8	638976
2	1090	180	28	769976
3	1545	90	31	851736
4	2454	408	40	1197348
5	3334	668	72	3347780
6	4909	818	80	4620556
7	6026	802	88	5489252
8	7875	750	99	5874944
9	10500	1000	114	8196312
10	12364	728	131	8309308

5 Complexity Analysis

The complexity of the proposed algorithm depends upon the number of distinct attribute values for different attributes. Total computational complexity is the sum of complexities of all the levels of the tree. There are two different cases (as discussed below) to calculate the complexity that 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, o, s, c, are the number of distinct attribute values for action, object, subject and contextual attribute values respectively. 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.

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

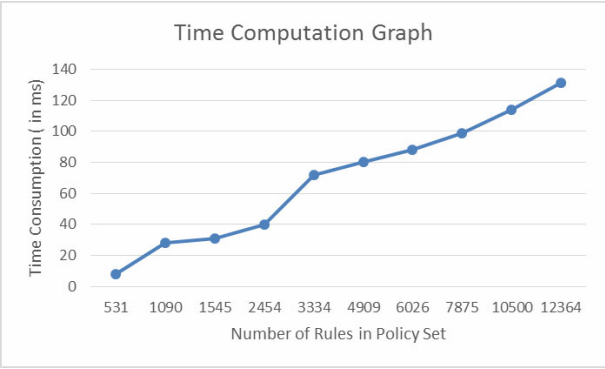

Figure 10. Time complexity analysis by running ACPs Validation Suite

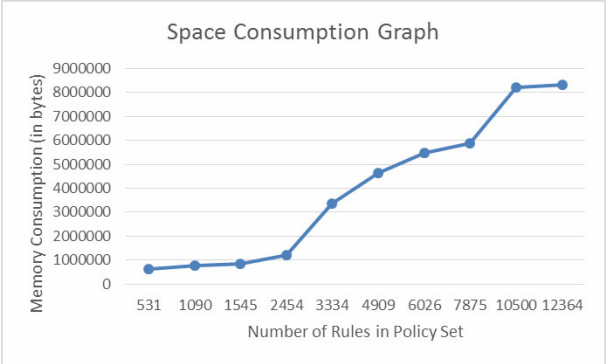

Figure 11. Space complexity analysis by running ACPs Validation Suite

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.

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

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 Figure 12 and Figure 13, we have shown the complexity of proposed algorithms for case 1 and case 2 respectively. 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.

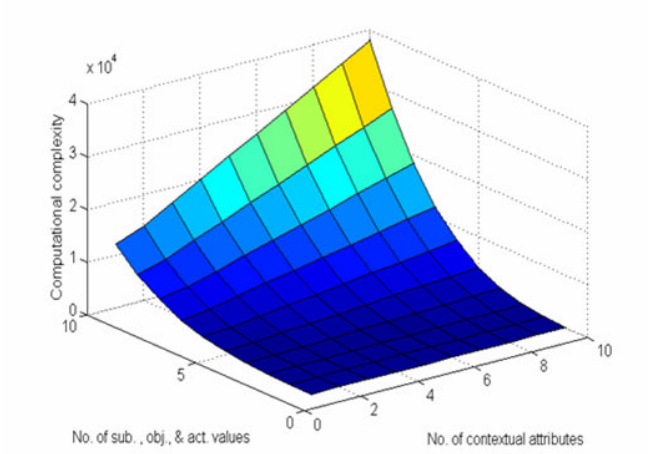

Figure 12. Complexity analysis of proposed algorithm for Case 1

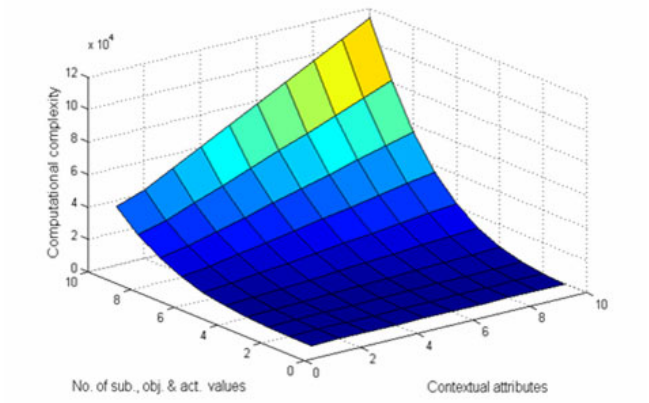

Figure 13. Complexity analysis of proposed algorithm for Case 2

6 Related Work and Qualitative Comparison

6.1 Related Work

Many policy validation methods have been presented by researchers for the verification and validation of access control policies. Various approaches have been used to detect inconsistencies in access control policies such as modeling techniques, formal methods, data mining techniques and classification algorithms etc. For more details, readers can consult the survey paper by Aqib and Shaikh [1].

Sheikh *et al.* [2] and Fisler *et al.* [10] have used the decision diagram techniques for this purpose. Sheikh *et al.* have used the data classification techniques like ID3 [11], C4.5 [12] and ASSISTANT 86 [13] to generate decision trees. They have proposed some modifications in these algorithms. On the other hand, Fisler *et al.* have used another type of decision diagram techniques and have presented a software called Margrave by implementing the binary decision diagrams. It also includes the rule-combining algorithms of XACML [21].

Mukkamala *et al.* [14], Bauer *et al.* [9] and Evan Martin and Tao Xie [15] have used the data mining approach to resolve this issue in access control policies. They have used the techniques, mainly used in data mining for the extraction of required data from the large amount of data to detect the rules in defined policies that make those policies inconsistent. For example, Bauer *et al.* [9] have used the Apriori algorithm [16] by the authors to apply the association rule mining approach.

Modeling tools have also been used by researchers for the verification and validation of access control policies. Different modelling languages have used for this purpose. For example, Hwang *et al.* [16] have developed a tool named Access Control Policy Testing (ACPT) and have used the symbolic model checker NuSMV [17]. Similarly, Mankai and Logrippo [4] also have proposed a solution for this purpose and they also have used a modeling tool Alloy [18-20].

6.2 Qualitative Comparison with Existing Methods

In Table 4, we have compared our scheme with 12 other existing validation methods. The comparison has been presented from the following seven parameters:

(1) Approach
(2) Inconsistency detection
(3) Inconsistency resolution
(4) Boolean expression
(5) Continuous attribute handling
(6) Dynamic data handling
(7) Contextual attributes

Table 4 clearly indicates that our tool has its distinctive place in state-of-the-art work. In terms of feature comparison, our work is similar to the work of Shaikh *et al.* [2-3]. However, they have adopted data classification approach and we have adopted tree-based algorithmic approach. On the positive side, our method is relatively easy to implement and we have provided proof-of-concept implementation. On the negative side, our work is limited to inconsistency detection only. Whereas, Shaikh *et al.* [3] shows that their method can also be used to detect incompleteness.

Table 4. Qualitative comparison of different policy validation techniques

	Approach	Inconsistency Detection	Inconsistency Resolution	Boolean Expression	Continuous Data Handling	Dynamic Data Handling	Contextual Attributes
Our Proposed Method	Decision Tree based Algorithm	Yes	No	Yes	Yes	Yes	Yes
Shaikh et al. [2]	Data classification	Yes	No	Yes	Yes	Yes	Yes
Mankai and Logrippo [4]	Model checking Alloy	Yes	No	No	No	No	No
Bei et al. [7]	Matrix based algorithm	Yes	No	Yes	Yes	No	Yes
Karimi and Cowan [5]	Model checking Alloy	Yes	Yes	No	No	No	No
Ma et al. [6]	Model Checking SPIN	Yes	No	No	No	No	No
Sun et al. [22]	Purpose based access control model	Yes	No	Yes	No	No	Yes
Abbasi and Fatmi [23]	Promela specification language, RG	Yes	No	No	No	No	No
Bravo et al. [24]	DTD graph, algorithms	Yes	Yes	No	Yes	No	No
Shafiq et al. [25]	Integer Programming technique, graphs, algorithm	Yes	Yes	No	Yes	No	No
Fisler et al. [10]	Decision diagrams MTBDD	Yes	Yes	Yes	No	No	No
Bauer et al. [9]	Association rule mining approach	Yes	Yes	Yes	No	Yes	No
Martin and Xie [15]	Data Mining Approach	Yes	No	No	No	No	No

7 Conclusion and Future Work

In this article, we have proposed an algorithm-based approach to detect inconsistencies in the access control policies. Also, we have developed a tool to validate the access control policies by implementing the proposed algorithm. We demonstrate that our proposed can efficiently detect inconsistencies in access control policies especially those which involve contextual attributes and Boolean expressions. By supporting Boolean expressions, continuous attribute values, and contextual attribute values, our proposed algorithm also reduces the number of rules.

The proposed solution 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. So in the future, we are planning to address these issues. In addition to these, we will also improve the performance in terms of computational complexity.

References

[1] M. Aqib, R. A. Shaikh, Analysis and Comparison of Access Control Policies Validation Mechanisms, International Journal of Computer Network and Information Security, Vol. 7, No. 1, pp. 54-69, February, 2015.

[2] R. A. Shaikh, K. Adi, L. Logrippo, S. Mankovski, Inconsistency Detection Method for Access Control Policies, *Proc. of Sixth International Conference on Information Assurance and Security*, Atlanta, Georgia, 2010, pp. 204-209.

[3] R. A. Shaikh, K. Adi, L. Logrippo, A Data Classification Method for Inconsistency and Incompleteness Detection in Access Control Policy Sets, *International Journal Information Security*, Vol. 16, No. 1, pp. 91-113, February, 2017. doi:10.1007/s10207-016-0317-1

[4] M. Mankai, L. Logrippo, Access Control Policies: Modeling and Validation, *Proc. Of the 5th NOTERE Conference*, Gatineau, Canada, 2005, pp. 85-91.

[5] V. R. Karimi, D. D. Cowan, Verification of Access Control Policies for REA Business Processes, *33rd Annual IEEE International Computer Software and Application Conference*, Seattle, Washington, 2009, pp. 422-427.

[6] J. Ma, D. Zhang, G. Xu, Y. Yang, Model Checking Based Security Policy Verification and Validation, *Proc. of 2nd International Workshop on Intelligent Systems and Applications (ISA), IEEE*, Wuham, China, 2010, pp. 1-4.

[7] B. Wu, X. Chen, Y. Zjang, X.-D. Dai, An Extensible Intra Access Control Policy Conflict Detection Algorithm, *Proc. of International Conference on Computational Intelligence and Security,* San Francisco, CA, 2009, pp. 483-488.

[8] A. Mohan, D. M. Blough, T. Kurc, A. Post, J. Saltz, Detection of Conflicts and Inconsistencies in Taxonomy-based Authorization Policies, *Proc. of IEEE International Conference on Bioinformatics and Biomedicine*, Atlanta, Georgia, 2011, pp. 590-594.

[9] L. Bauer, S. Garriss, M. K. Reiter, Detecting and Resolving Policy Misconfigurations in Access-Control Systems, *Proc. of ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008, New York, NY, pp. 185-194.

[10] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, M. C. Tschantz, Verification and Change-impact Analysis of Access-Control Policies, *Proc. of the 27th International Conference on Software Engineering*, New York, NY, 2005, pp. 196-205.

[11] Quinlan, J. R. Induction of Decision Trees, *Mach. Learn*, Vol. 1, No. 1, pp. 81-106, March, 1986.

[12] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

[13] B. Cestnik, I. Kononenko, I. Bratko, Assistant 86: A Knowledge Elicitation Tool for Sophistical Users, *Proc. of*

the 2nd European Working Session on Learning, Sigma, Wilmslow, 1987, pp. 31-45.

[14] R. Mukkamala, V. Kamisetty, P. Yedugani, Detecting and Resolving Misconfigurations in Role-Based Access Control, *Proc. of 5th International Conference on Information System Security (ICISS)*, Kolkata, India, 2009, pp. 318-325.

[15] E. Martin, T. Xie, Inferring Access-Control Policy Properties via Machine Learning, *Proc. of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*, Washington, DC, 2006, pp. 235-238.

[16] J. Hwang, T. Xie, V. Hu, M. Altunay, ACPT: A Tool for Modeling and Verifying Access Control Policies, *Proc. of IEEE International Symposium on Policies for Distributed Systems and Networks,* Fairfax, VA, 2010, pp. 40-43.

[17] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking, *Proc. of 14th International Conference on Computer Aided Verification (CAV)*, Copenhagen, Denmark, 2002, pp. 359-364.

[18] D. Jackson, *ALLOY Home Page*, http://sdg.lcs.mit.edu/alloy/detaulb.htm

[19] D. Jackson, *Micromodels of Software: Lightweight Modelling and Analysis with ALLOY*, http://sdg.lcs.mit.edu/dng

[20] D. Jackson, *ALLOY 3.0 Reference Manual*, http://alloy.mit.edu/beta/reference-manual.pdf

[21] E. Rissanen, *eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard*, http://docs.oasis-open.org/xacml/3.0/ xacml-3.0-core-spec-os-en.html

[22] L. Sun, H. Wang, X. Tao, Y. Zhang, J. Yang, Privacy Preserving Access Control Policy and Algorithms for Conflicting Problems, *Proc. of International Joint Conference of IEEE TrustCom,* 2011, pp. 250-257.

[23] R. Abassi, S. Guemara, E. Fatmi, An Automated Validation Method for Security Policies: The Firewall Case, *Proc. of the 4th International Conference on Information Assurance and Security*, Napoli, Italy, 2008, pp. 291-294.

[24] L. Bravo, J. Cheney, I. Fundulaki, ACCOn: Checking Consistency of XML Write-Access Control Policies, *Proc. of the 11th Int. Conference on Extending Database Technology: Advances in Database Technology*, Nantes, France, 2008, pp. 715-719.

[25] B. Shafiq, J. Vaidya, A. Ghafoor, E. Bertino, A Framework for Verification and Optimal Reconfiguration of Event-driven Role Based Access Control Policies, *Proc. of ACM Symp. on Access Control Models and Technologies (SACMAT)*, Newark, NJ, 2012, pp. 197-208.

[26] M. Guarnieri, M. Marco, Arrigoni Neri, E. Magri, S. Mutti, On the Notion of Redundancy in Access Control Policies, *Proce. of the 18th ACM Symposium on Access Control Models and Technologies*, New York, NY, 2013, pp. 161-172.

Biographies

Muhammad Aqib is currently working towards the Ph.D. degree in HPC and Big Data at King Abdulaziz University, Jeddah, Saudi Arabia. He received the M.S. degree in Computer Science from KAU in 2014. His research interest includes HPC, big data, privacy, and security.

Riaz Ahmed Shaikh is an Associate 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.