Security Enhancement in Web Services by Detecting and Correcting Anomalies in XACML Policies at Design Level

M. Priyadharshini[1], J. Yowan[2], and R. Baskaran[1]

[1] Computer Science Department, Anna University, India
mpriya1977@gmail.com, baaski@annauniv.edu
[2] Department of Information Science and Technology, Anna University, India
jyowan@gmail.com

Abstract. The significance of XACML (Extensible Access Control Markup Language) policies for access control is immeasurably increasing particularly in web services. XACML policies are web access control policies which are used to permit the genuine users to access the resources and also deny the sham users. Generation of this XACML policy is very important task in order to avoid security seepage. Detecting and Correcting inconsistencies in access control policies are highly time consuming and tedious when size of XACML polices are high. The Process when done at execution time could even need more time and effort. The purpose of this work is to devise an anomaly detection and correction tool which could be used at the time of designing policies so as to reduce time and effort. Policy designer could easily discover and resolve the inconsistencies such as conflicts and redundancies in the XACML policies with the help of our XACML Policy Analyzer tool.

Keywords: Web Services, XACML, Access Control, Anomalies, Policy Analyzer.

1 Introduction

Extensible Access Control Markup Language(XACML) [1], which is a general purpose access control policy language drafted by the Organization for the Advancement of Structured Information Standards (OASIS), has been broadly used to specify access control policies for numerous applications, especially in the domain of Web Services [3]. XACML policy designer when creating XACML Policies may sometimes create it with inconsistencies [2]. Conflict and Redundancy are the two types of inconsistency that exist in XACML Policies. In the XACML policies redundancies occur, if one rule's content may be repeated in another rules with same effect such as permit or deny. Conflicts occur, if one rule's content may be repeated in other rules with different effect such as permit or deny. Redundancies in the policies increase the policy evaluation time. Conflicts in XACML Policies lead two kinds of problems. First, Security problem (e.g. permitting sham users).Second, accessibility problem (e.g. denying genuine users).The Policy designer can manually resolve these problems only for the policies which contain few rules. The Policy designer cannot manually resolve these problems for the policies which contain many rules.

XACML has four different combining algorithms [1] such as Deny-overrides, Permit-overrides, First-Applicable and Only-One-Applicable. Policy Designer cannot assign correct combining algorithm without correct conflict information about the XACML policies. Redundancy elimination is the solution for XACML policy optimization. Policy optimization can improve the performance of XACML Evaluation.

In this paper, we propose four algorithms for detecting and correcting anomalies in XACML policies. First, Redundancy Detecting Algorithm which finds the redundancies present in the policies. Second is the Redundancy Elimination Algorithm that eliminates the redundancies in the policies. Third, Conflict Detection Algorithm is used to identify the conflicts in the policies. Fourth, Conflict Correction Algorithm is used to resolve the conflicts in the policies.At present XACML conflict correction mechanisms use only one combining algorithm to resolve all detected conflicts within an XACML policy or policy set. Also, many other conflict correction methods are present [4], [5], [6], but they don't support XACML. Thus we provide a policy detection and conflict correction mechanism for XACML in this proposed work.

The rest of this paper is organized as follows. Section 2 describes the overview of XACML Policy and a discussion on the anomalies in the XACML Policy is also been presented. Section 3 presents the proposed methodology. In Section 5, we present the results and discussion of the proposed methodology. In Section 6 few related works are specified and finally conclusion and future work in Section 7.

2 Overview of XACML

The root of all XACML policies is PolicySet or a Policy. A PolicySet contains many Policies or other PolicySets and corresponding policy combining algorithm. A Policy contains a target, set of rules and also rule combining algorithm. The target defines a set of subjects, resources and actions. A rule set is a sequence of rules. Each rule consists of a target, a condition, and an effect.

The target of a rule determines whether an access request is applicable to the rule and it has a similar structure as the target of a policy or a policy set. Table 1 shows a sample XACML policy. The root policy set **pset1** contains two policies, **Pol1** and **Pol2**, which are combined using First-Applicable combining algorithm. The policy **Pol1** has two rules, **r1** and **r2**, and its rule combining algorithm is Deny-Overrides. The policy **Pol2** includes four rules **r3**, **r4**, **r5** and **r6** with Deny-Overrides combining algorithm. In this sample, there are three subjects: Assistant **Manager**, **Project Manager**, and **Site Engineer**; two resources: **Project Report** and **Contractor Details**; and two actions: **Read** and **Write**; one condition with time attribute: **8<=Time<= 18**.

Table 1. Sample XACML policy

```
<PolicySetPolicySetid=”pset1” Policy Combining AlgId=”First-Applicable”>
 <Target/>
  <Policy PolicyId=”pol1”   RuleCombiningAlgId=”Deny-Overrides”>
   <Target/>
    <Rule RuleId=”r1”  Effect=”Permit”>
     <Target>
      <Subjects>
        <Subject>Project Manager</Subject>
      </Subjects>
      <Resources>
         <Resource>Project Report</Resource>
         <Resource>Contractor details</Resource>
      </Resources>
     <Actions>
         <Action>Read<Action>
         <Action>Write<Action>
      </Actions>
     </Target>
  </Rule>
  <Rule RuleId=”r2”  Effect=”Permit”>
      <Target>
        <Subjects>
              <Subject>Project Manager</Subject>
              <Subject>Assistant Manager</Subject>
        </Subjects>
        <Resources>
              <Resource>Project Report</Resource>
              <Resource>Contractor details</Resource>
        </Resources>
        <Actions>
              <Action> Read<Action>
        </Actions>
      </Target>
     </Rule>
 </Policy>
<Policy PolicyId=”pol2” RuleCombiningAlgId=”Permit-Overrides”>
  </Target>
     <Rule RuleId=”r3”  Effect=”Permit”>
       <Target>
         <Subjects>
               <Subject>Site Engineer</Subject>
         </Subjects>
        <Resources>
               <Resource>Contractor details</Resource>
        </Resources>
        <Actions>
                <Action> Read<Action>
        </Actions>
      </Target>
        <Condition>8 <=Time<=18</Condition>
      </Rule>
      <Rule Rule Id=”rule4”  Effect=”Permit”>
        <Target>
          <Subjects>
               <Subject>Assistant Manager</Subject>
          </Subjects>
          <Resources>
               <Resource>Project Report</Resource>
          </Resources>
          <Actions>
               <Action>Read<Action>
          </Actions>
        </Target>
      </Rule>
      <Rule Rule Id=”r5”  Effect=”Deny”>
       <Target>
          <Subjects>
               <Subject>Site Engineer</Subject>
         </Subjects>
         <Resources>
               <Resource>Contractor details</Resource>
             </Resources>
             <Actions>
                    <Action>Read<Action>
             </Actions>
          </Target>
        </Rule>
        <Rule Rule Id=”r6”  Effect=”Deny”>
          <Target>
              <Subjects>
                   <Subject>Assistant Manager</Subject>
              </Subjects>
              <Resources>
                   <Resource>Project Report</Resource>
              </Resources>
              <Actions>
                   <Action>Read<Action>
              </Actions>
            </Target>
         </Rule>
      </Policy>
  </PolicySet>
```

2.1 Anomalies in XACML

Redundancy and Conflict are two anomalies that could exist between Policies and Policy Sets which need to be resolved so as to enable taking smooth decision in case of access control.

Anomalies at Policy Level

A Policy contains a target, set of rules and also rule combining algorithm and there is a possibility that two rules defined may lead to different decisions that is to permit or deny or else even they may have same redundant rule been defined thus leading to anomalies at Policy level.

Redundancies:

In Policy pol1, rule r1 is in redundant with rule r2 in Fig 1 where r1 allows the Project Manager (Subject) to read (Action) Project Report and Contract details (Resources) which is also permitted by r2.The complete set of redundancies in each policy of the sample XACML policy in Table 1 is shown in Table 2.

Table 2. Redundancy at Policy Level

S. no	*Policy Id*	*Redundancy between the rules*				
1	pol1	**Rule Id**	**r1**	**r2**		
		r1	-	yes		
		r2	yes	-		
2	pol2					
		Rule Id	**r3**	**r4**	**r5**	**r6**
		r3	-	no	no	no
		r4	no	-	no	no
		r5	no	no	-	no
		r6	no	no	no	-

Conflicts:

*In Policy **pol2**, rule **r3** is in conflict with rule **r5** and also **r4** is in conflict with **r6** in **Table 1** because **r3** allows the Subject **Site Engineer** to **read** (Action) the Resource **Contract details** in the time interval [8:00, 18:00] which are denied by **r5.** The complete conflicts in each policy of the sample XACML policy in Fig 1 is shown in Table 3.*

Anomalies at Policy Set Level

A Policy Set has a set of policies and there is a possibility that two policies defined could possess same target, set of rules and rule combining algorithm leading to different decisions that is to permit or deny or else even they may have same redundant policy been defined thus leading to anomalies at policy set level.

Table 3. Coflicts at policy level

S. no	*Policy Id*	*Conflicts between the rules*				
1	pol1	**Rule Id**	**r1**	**r2**		
		r1	-	no		
		r2	no	-		
2	pol2					
		Rule Id	**r3**	**r4**	**r5**	**r6**
		r3	-	no	yes	-
		r4	-	-	-	yes
		r5	yes	-	-	-
		r6	-	yes	-	-

Redundancies:

Redundancies may also occur between policies or policy sets. Rule ***r2*** *of Policy* ***pol1*** *is in redundant with rule* ***r4*** *of Policy* ***pol2*** *in Table 1because* ***r2****allows the* ***Assistant Manager*** *(Subject) to* ***read*** *(Action)* ***Project Report*** *(Resource) which is also permitted by* ***r4****. The complete redundancies between each policies of the sample XACML policy in Table 1 is shown in Table 4.*

Table 4. Redundancy in policy set level

S. no	*Policy Ids*	*Redundancy between the rules*				
		Rule Id	**r3**	**r4**	**r5**	**r6**
1	pol1 and pol2	**r1**	no	no	no	no
		r2	no	yes	no	no

Conflicts:

Conflicts may also occur between policies or policy sets. Rule ***r2*** *of Policy* ***pol1*** *is in conflict with rule* ***r6*** *of Policy* ***pol2*** *in Table 1 because* ***r2*** *allows the* ***Assistant Manager*** *(Subject) to* ***read*** *(Action)* ***Project Report*** *(Resource) which is denied by* ***r6****. The complete conflicts between each policies of the sample XACML policy in Fig 1 is shown in Table 5.*

Table 5. Conflicts at policy set level

S. no	*Policy Ids*	*Conflicts between the rules*				
		Rule Id	**r3**	**r4**	**r5**	**r6**
1	pol1 and pol2	**r1**	no	no	No	no
		r2	no	no	No	yes

3 Proposed Methodology

The proposed methodology consists of sequence of activities to construct the Boolean Expression followed by the Algorithms to detect as well as to correct the anomalies in the XACML Policy. The algorithms include redundancy detection algorithm, redundancy elimination algorithm, conflict detection algorithm and finally conflict correction algorithm as given in Fig. 1 below.

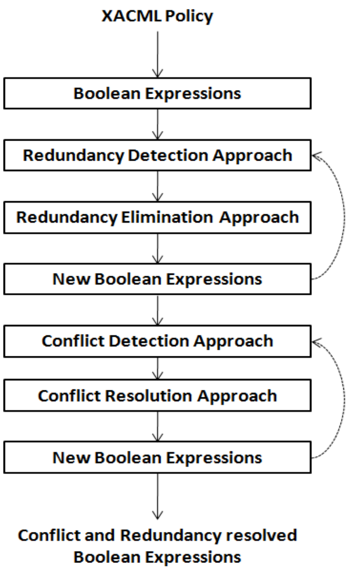

XACML Policy
Boolean Expressions
Redundancy Detection Approach
Redundancy Elimination Approach
New Boolean Expressions
Conflict Detection Approach
Conflict Resolution Approach
New Boolean Expressions
Conflict and Redundancy resolved Boolean Expressions

Fig. 1. Proposed Architecture

3.1 Construction of Boolean Expression

First, XACML policy is parsed to identify policy set id, policy id, rule effect, rule id and also attributes such as subjects, actions, resources and conditions. The parsed result is used to form Boolean Expression.

Second, After Parsing, a new Boolean variable should be assigned to each attributes. We assign each of the attributes values as a Boolean variable. For example, an attribute Subject="Project Manager" is assigned into a Boolean variable S1.

Table 6. Boolean Expressions of XACML Policy in Table 1

S. no	*Rule*	*Effect*	*Boolean Expression*
1	r1	Permit	$(S1) \wedge (R1 \vee R2) \wedge (A1 \vee A2)$
2	r2	Permit	$(S1 \vee S2) \wedge (R1 \vee R2) \wedge (A1)$
3	r3	Permit	$(S3) \wedge (R2) \wedge (A1) \wedge (C1)$
4	r4	Permit	$(S2) \wedge (R1) \wedge (A1)$
5	r5	Deny	$(S3) \wedge (R2) \wedge (A1)$
6	r6	Deny	$(S2) \wedge (R1) \wedge (A1)$

Third, we utilize the Variable Assigning technique to construct Boolean expressions in terms of Boolean variables for XACML rules. Each Boolean expression of a rule contains attributes combined by logical operator ∨and ∧. Boolean Expressions are used for anomaly detection, elimination and correction. Thus Table 6 shows the complete list of Boolean expressions for the XACML in Table 1.

3.2 Redundancy Detection Algorithm

Redundancy Detection Algorithm is used to find the redundancies present in the XACML Policies. Line 4-15 in Algorithm 1 contains redundancy_detection() method which can find the redundancies in the XACML Policies. Redundant rules have the following two properties:

1. All rules are pair wise disjointed
   Rule id $r_i \neq$ Rule id r_j
2. The effects of matched rules contain either “Permit” only or “Deny” only.

Line7 in Algorithm1 checks these properties. In **Table 6**, rule **r1** is in redundant with rule **r2** because Subject (S1), Resources (R1, R2), and Action (A1) are present in both the rules. So Subject_Match_r_i_r_j, Resource_Match_r_i_r_j, Action_Match_r_i_r_j and No_Condition_r_i_r_j from **Line 8-11** in Algorithm 1 are greater than zero.

Algorithm1. Redundancy Detection in XACML Policy

Input: Boolean Expression of rules with Effect

Output: Detected Redundancies between the rules

```
1    /*parsing the XACML Policy*/
2    /*Boolean Encoding*/
3    /*Boolean Expression of each rules*/
4    redundancy_detection()
5       for each $r_i \in B_e$
6         for each $r_j \in B_e$
7           if $\exists\ r_i \in B_e,\ r_j \in B_e, r_i \neq r_i, r_i$.Effect=$r_j$.Effect
8             if Subject_Match_$r_i$_$r_j$>0
9               if Resource_Match_$r_i$_$r_j$>0
10                if Action_Match_$r_i$_$r_j$>0
11                  if ConditionMatch_of_ $r_i$_$r_j$>0
12                  ||No_Condition_Match_$r_i$_$r_j$=0
13                    Redundancy present between $r_i$ and $r_j$
14                    redundancy_elimination();
15                    new_boolean_expression();
16                    redundancy_detection();
```

3.3 Redundancy Elimination Algorithm

After finding the redundancy, Redundancy Elimination algorithm identifies the redundancy type so as to find the different set of solutions to handle the redundancy.

Check Points for Redundancy Elimination

Redundancy Elimination Algorithm consist four Check points. The Check Points are Same, Different Equal, Different Subset and Different Superset. These Check Points analyse and find out the correct redundancy type. The Check Points for the redundancy elimination approach are shown in Fig 2.

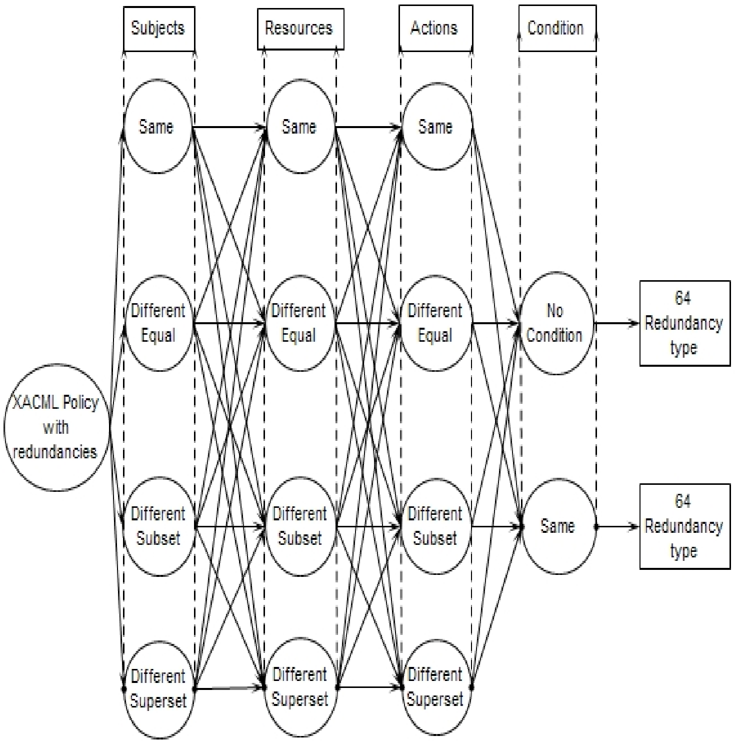


Fig. 2. Check Points for Redundancy Elimination

The Check points are briefly explained below:

Same: All the attribute values of same attribute group for rule r_i and r_j should be same. Consider rule r1 and r2 from the Boolean Expression Table, Here Resources R1 and R2 are present in both the rules. All Resources of r1 are not only same to the Resources of r2 but also the Resource length of r1 is equal to the Resource length of r2.Here Resource length of r1 and r2 is 2.

Different Equal: Few attribute values of same attribute group for rule r_i and r_j should be same and also the length of the attribute groups of the both the rules r_i and r_j should be same. Consider rule r1 and r2 from the Boolean Expression Table, For instance, if we change the Resource R2 of r2 to R3, now the Resources of rule r1 are R1and R3.Here only Resource R1 is present in both the rules. Few resources of r1 are not only same to the Resources of r2 but also the Resources length of r1 is equal to the Resource length of r2.Here Resource length of r1 and r2 is 2.

Different Subset: Few attribute values of same attribute group for rule $\mathbf{r_i}$ and $\mathbf{r_j}$ should be same and also the attribute group length of the rule $\mathbf{r_i}$ should be less than $\mathbf{r_j}$. Consider rule **r1** and **r2** from the Boolean Expression Table. Here Subject S1 is present in both the rules. Subject **S2**of **r2** is not present in the rule **r1**.Few Subjects of **r1** are not only same to the Subjects of rule **r2** but also the Subjects length of **r1** is lesser than the Subjects length of **r2**.Here Subject length of **r1** is **1** and Subject length **r2** is **2**.

Different Superset: Few attribute values of same attribute group for rule $\mathbf{r_i}$ and $\mathbf{r_j}$ should be same and also the attribute group length of the rule $\mathbf{r_i}$ should be greater than $\mathbf{r_j}$. Consider rule **r1** and **r2** from the Boolean Expression Table. Here Actions **A1** and **A2**are present in both the rules. Action **A2**of **r1** is not present in the rule **r2**.Few Actions of **r1** are not only same to the Actions of rule **r2** but also the Actions group length of **r1** is greater than the **r2**.Here Action group length of **r1** is **2** and length of **r2** is **1**.

The function redundancy_detection() in Algorithm 1 is invoked to find the redundancy between the rules. In Boolean Expression Table 6, rule **r1** is redundant with rule **r2**.The function redundancy_elimination() in Algorithm 2 uses Check point to eliminate the redundancy between the rules. The Check Points between these rules are Different Subset for Subjects, Same for Resources and Different Superset for Actions. The Solution for this redundancy type is to remove the Subject **S1** from the rule **r2**.Now new_boolean_expression() function in Algorithm 1 is called to assign new Boolean expression for rule **r1** and **r2**.The Boolean expression after redundancy elimination is shown in the Table 7**.**

Table 7. Boolean Expression after Redundancy Elimination

S. no	*Rule*	*Effect*	*Boolean Expression*
1	r1	Permit	$(S1) \wedge (R1 \vee R2) \wedge (A1 \vee A2)$
2	r2	Permit	$(S2) \wedge (R1 \vee R2) \wedge (A1)$
3	r3	Permit	$(S3) \wedge (R2) \wedge (A1) \wedge (C1)$
4	r4	Permit	$(S2) \wedge (R1) \wedge (A1)$
5	r5	Deny	$(S3) \wedge (R2) \wedge (A1)$
6	r6	Deny	$(S2) \wedge (R1) \wedge (A1)$

Again redundancy_detection() function is called to find the redundancy between the rules in the new Boolean expression. Now rule **r2** is redundant with rule **r4**. Again function redundancy_elimination() uses Check points to eliminate the redundancy between the rules. The Check Points between these rules are Same for Subjects, Different Superset for Resources and Different Superset for Actions. The Solution for this redundancy type is to remove the rule **r4** from the Sample XACML Policy. Now new_boolean_expression() function assign new Boolean expression for the new Sample Policy. This process continues until eliminate the redundancies in the XACML Policy completely. A complete list of Boolean encoding for the example XACML policy in Table 1 is shown in Table 2.

3.4 Conflict Detection Algorithm

Conflict Detection Algorithm is used to find the conflicts present in the XACML Policies. Line 4-25 contains conflict_detection() method which can find the conflicts in the XACML Policies.

Algorithm2. Conflict Detection in XACML Policy

Input: New Boolean Expression without redundancy

Output: Detect Conflicts present between the rules

```
1    /*parsing the XACML Policy*/
2    /*Boolean Encoding*/
3    /*Boolean Expression of each rules*/
4    conflict_detection()
5        for each r ∈ $nB_e$
6          for each r ∈ $nB_e$
7            if ∃ $r_i∈nB_e$, $r_j∈nB_e$,$r_i≠r_j$,$r_i$.Effect≠t$r_j$.Effect
8                if Subject_Match_$r_i$_$r_j$,>0
9                  if Resource_Match_$r_i$_$r_j$,>0
10                   if Action_Match_ $r_i$_$r_j$,>0
11                       if Condition_Match_$r_i$_$r_j$,>0
12                       ||No_Condition_Match_$r_i$_$r_j$=0
13                       ||One_Condition_$r_i$_$r_j$>0
14                         Conflict present between $r_i$ and $r_j$
15           if ∃ $r_i$∈ Ps, $r_j$∈ Ps,$r_i≠r_j$,$r_i$.Effect=$r_j$.Effect
16               if Subject_Match_$r_i$_$r_j$,>0
17                 if Resource_Match_$r_i$_$r_j$,>0
18                   if Action_Match_$r_i$_$r_j$,>0
19                       if One_Condition_$r_i$_$r_j$>0
20                         Conflict present between $r_i$ and $r_j$
21
22
23                             get_policy_designer_effect();
24                             conflict_resolution();
25                             new_boolean_expression();
```

Conflicting rules have the following two Conditions:
Condition1 has three properties

1. All rules are pair wise disjointed
   Rule id $r_i \neq$ Rule id r_j
2. The effects of matched rules should contain both "Permit" and "Deny"
3. If Condition attribute present in the rules then
   "NoCondition list length is greater than zero" or
   "ConditionMatch list length is greater than Zero"
   or "OneCondition list length is greater than Zero".

Line7-11 in the Algorith 2 checks these properties.
Condition2 has three properties

1. All rules are pair wise disjointed
   Rule id $r_i \neq$ Rule id r_j
2. The effects of matched rules contain either "Permit" only or "Deny" only.
3. If condition attribute present in the rules then "OneCondition list length should be greater than Zero".

Line17-21 of Algorithm 2 checks these properties.

In new Boolean Expression table, rule **r3** is in conflict with rule **r5** because Subject (S3), Resources (R2), and Action (A1) are present in both the rules and also Condition attributes is present only in the rule **r3**. So Subject_Match_r_i_r_j, Resource_Match_r_i_r_j, Action_Match_r_i_r_j and One_Condition_r_i_$r_j$size are greater than zero. Rule **r2** is in conflict with rule **r6**becauseSubject (S1), Resources (R1), and Action (A1) are present in both the rules and also So Subject_Match_r_i_r_j, Resource_Match_r_i_r_j, and Action_Match_r_i_$r_j$sizeare greater than zero.

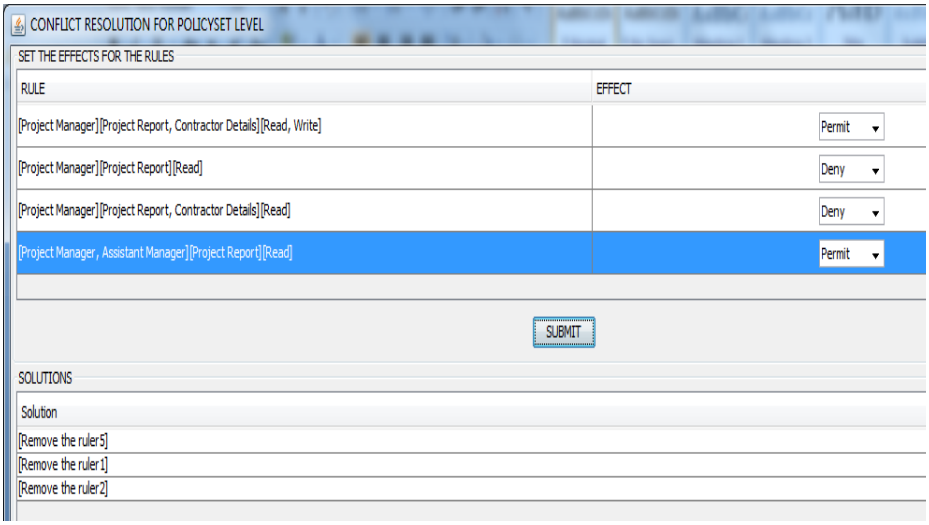
3.5 Conflict Resolution Algorithm

First, the function conflict_detection() is called to find the conflicts between the rules. In Boolean Expression table, rule **r2** is in conflict with rule**r6**and also **r3** is in conflict with **r5**.

Second, get_policy_designer_effect() is called to get the rule effects of the conflicting rules and show that effects and conflicting segments to the Policy Designer. The policy designer should set the correct effects to those conflicting segments; to enable easy interpretation a grid based representation is used. Here the following three segments are the conflicting segments.

1. S2-R1-A1 from rule r2
2. S3-R2-A1-C1 from rule r3
3. S3-R2-A1 from rule r5

If the Policy Designer set the effect **Permit** to the conflicting segments S2-R1-A1 and S3-R2-A1-C1, the conflicting segment S3-R2-A1 will be hidden to the Policy designer for which the system provides a user interface to suggest the policy designer as in Fig. 3.


CONFLICT RESOLUTION FOR POLICYSET LEVEL
SET THE EFFECTS FOR THE RULES
SUBMIT
SOLUTIONS

RULE	EFFECT
[Project Manager][Project Report, Contractor Details][Read, Write]	Permit
[Project Manager][Project Report][Read]	Deny
[Project Manager][Project Report, Contractor Details][Read]	Deny
[Project Manager, Assistant Manager][Project Report][Read]	Permit

Solution
[Remove the ruler5]
[Remove the ruler1]
[Remove the ruler2]

Fig. 3. User Resolution suggestion User Interface

Third, the function conflict_resolution() uses Check point to resolve the conflicts between the rules r2 and r6 and also between the rule r3 and r5. The Check Points between the rule r2 and r6 are Same for Subjects, Different Superset for Resources and Same for Actions. The Solution for this conflict type is to remove the rule **r6**.The Check Points between the rule **r3** and **r5** are Same for Subjects, Same for Resources, Same for Actions and one condition is present in **r3**. The Solution for this conflict type is to remove the rule **r5**.After the conflict resolution new_boolean_expression() function assign new Boolean expression we get new Refined Boolean Expression presented in Table 8.

Table 8. Refined Boolean Expression

S. no	*Rule*	*Effect*	*Boolean Expression*
1	r1	Permit	$(S1) \wedge (R1 \vee R2) \wedge (A1 \vee A2)$
2	r2	Permit	$(S2) \wedge (R1 \vee R2) \wedge (A1)$
3	r3	Permit	$(S3) \wedge (R2) \wedge (A1) \wedge (C1)$

4 Results and Discussion

The proposed methodology of conflict and redundancy elimination covers the already prevailing methodologies where only part of the methodology was implemented and analyzed.

Table 9 presents various tools(T) and approaches(A) that are used for detecting and resolving anomalies in XACML Policy, the features they support and hence the purpose of the proposed methodology to highly reduce the processing time of PDP is also solved resolving the anomalies.

Table 9. Listing of Tools and Approaches for Anamoly Detection and Resolution

S. no	*Tools/Approach*	*Phase*	*Features Supported*	*Features Not Support*
1	Margrave(T)	Implementation	Policy Verification	Anomaly Resolution
2	EXAM(T)	Implementation	Policy Analysis, Policy Verification, Policy Integration	Anomaly Resolution
3	PCL(A)	Implementation	Conflict Resolution	Without pre-defined combining algorithm in XACML Policy
4	XEngine(T)	Implementation	Conflict Resolution	Without predefined combining algorithms in the policy
5	SunPDP(T)	Implementation	Conflict Resolution	Without predefined combining algorithms in the policy
6	XAnalyzer(T)	Design	Redundancy Elimination and Conflict Resolution	Without predefined combining algorithms in the policy
7	XACML Policy Analyzer(T)	Design	Redundancy Elimination and Conflict Resolution without pre-defined combining algorithms	-

A sample policy with 6 Rules, 2 Policies and 1 Policy Set is taken as a sample and executed in our system and compared with that of XAnalyzer[2].Our approach is found to consume lesser time for redundancy removal as well as for conflict detection when compared to the other approaches and also this proves to be efficient since done at design level itself. The result of the experimentation is provided as follows in Table 10.

The other important aspect that could justify our approach is that our approach eliminates the process of applying rule combining algorithms during processing as done with other approaches which could raise the performance increase in total processing time.

Table 10. Comparision of Approaches in Conflict Resoultion and Redundancy Elimination

S. no	*Tools/Approach*	*Phase*	*Conflict Resolution Time(s)*	*Redundancy Elimination Time (s)*
1	XAnalyzer(T)	Design	0.82	0.87
2	XACML Policy Analyzer(T)	Design	0.80	0.84

5 Related Works

In [7], Dan Lin et al proposed a policy similarity measure for XACML policy similarity analysis. In [8], Lin et al designed a tool EXAM can be used for policy property analysis, policy similarity analysis and policy integration. Fisler et al. [9] designed a tool Margrave, which can verify policy properties and perform change-impact analysis. During runtime Sun PDP [10] and XEngine [11] detect the conflicts and resolve the conflicts by applying predefined combining algorithms in the policy. In [2], Hongxin Hu et al developed a tool XAnalyzer which can be used for policy analysis at policy design time. XAnalyzer can identify all conflicts within a policy and help policy designers to select appropriate combining algorithms for conflict resolution. Bauer et al. [12] adopted a data-mining technique to eliminate contradictions occurring between access control policies. Our tool XACML Policy Analyzer also concentrates on policy analysis at policy design time. XACML Policy Analyzer can identify all conflicts within a policy and finally it selects first - Applicable combining algorithm to resolve the conflicts.

Few other approaches which were concentrating on formalizing XACML policies were provided [13] and [14]. In [13] a process algebra named as Communication Sequential Process(CSP) is provided to verify the policy's properties as well as to compare access control policies. In [14] Ahn et al does the formalization using a declarative programming known as Answer Set Programming (ASP) which enables ASP reasoners to perform policy evaluation. In either methods of formalization no guarantee for the complete elicitation of policy properties and policy verification is provided.

6 Conclusions

We have proposed a mechanism that facilitates systematic detection and resolution of XACML policy anomalies eliminating the need to construct BDD. An efficient anomaly detection and resolution is done at the design level hence reducing the processing time of the PDP and this could be helpful in cases of larger XACML policies with more number of rules, policies and policy set. This also aims at security in terms of access control and to increase availability of the service. As a future extension this could be implemented for XACML3.0 and also could be included with handling obligations and user defined functions.

References

1. Godik, S., Moses, T.: Extensible Access Control Markup Language (XACML). version 2.0, OASIS Standard (2005)
2. Hu, H., Ahn, G., Kulkarni, K.: Discovery and Resolution of Anomalies in Web Access Control Policies, p. 11 (2013)
3. XACML. OASIS XACML committee website (2011), `http://www.oasisopen.org/committees/xacml/`
4. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A logical language for expressing authorizations. In: IEEE Symposium on Security and Privacy, Oakland, CA, pp. 31–42 (May 1997)
5. Jin, J., Ahn, G., Hu, H., Covington, M., Zhang, X.: Patient-centric authorization framework for sharing electronic health records. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, pp. 125–134. ACM, New York (2009)
6. Li, N., Wang, Q., Qardaji, W., Bertino, E., Rao, P., Lobo, J., Lin, D.: Access control policy combining: theory meets practice. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, pp. 135–144. ACM (2009)
7. Lin, D., Rao, P., Bertino, E., Lobo, J.: An approach to evaluate policy similarity. In: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, pp. 1–10. ACM (2007)
8. Lin, D., Rao, P., Bertino, E., Li, N., Lobo, J.: Exam: A Comprehensive Environment for the Analysis of Access Control Policies. International Journal of Information Security 9(4), 253–273 (2010)
9. Hu, H., Ahn, E.: Enabling Verification and Conformance Testing For Access Control Model. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, pp. 195–204. ACM (2008)
10. `http://sunxacml.sourceforge.net`
11. Liu, A., Chen, F., Hwang, J., Xie, T.: XEngine: A Fast and Scalable XACML Policy Evaluation Engine. ACM SIGMETRICS Performance Evaluation
12. Bauer, L., Garriss, S., Reiter, M.: Detecting and Resolving Policy Misconfigurations In Access-Control Systems. ACM Transactions on Information and System Security (TISSEC) 1, 2–5 (2011)
13. Bryans, J.: Reasoning about XACML policies using CSP. In: Proceedings of the 2005 workshop on Secure Web Services, p. 35. ACM (2005)
14. Ahn, G., Hu, H., Lee, J., Meng, Y.: Representing and Reasoning about Web Access Control Policies. In: 34th Annual IEEE Computer Software and Applications Conference, pp. 137–146. IEEE (2010)