DESIGN AND EVALUATION OF XACML CONFLICT POLICIES DETECTION MECHANISM

Kamalbir Singh and Sarbjeet Singh

Computer Science and Engineering, Panjab University, Chandigarh, India
kamalbirsingh@hotmail.com, sarbjeet@pu.ac.in

ABSTRACT

The evolution of distributed computing technologies like grid computing, peer-to-peer computing, pervasive computing, ubiquitous computing, autonomic computing, cloud computing etc. has led to the development of complex virtual systems. These systems enable sharing of resources and services distributed over geographically dispersed, heterogeneous, autonomous administrative domains and allow one to efficiently perform a compute or storage intensive task by harnessing the features available over other domains. The resources and services provided by service providers are generally protected by complex access control policies. These access control policies are expressed using policy specification languages. One of the popular, exhaustive and feature rich access control policy specification language is XACML, which is an OASIS standard also. The XACML policies are specified either manually or using automated XACML policy specification tools. Among major problems that policy administrators face is the problem of conflict policies. Conflict policies can have serious consequences and may lead to unauthorized access. This paper presents the design, implementation and evaluation of a conflict policy detection mechanism that can be used by policy administrators to proactively detect conflict XACML policies present in a policy database. This saves administrators and initiators of job from unnecessary problems arising due to presence of conflicts. The mechanism presented is simple, scalable, efficient and can be used to detect policies conflicting with respect to subject, resource, and action attributes. The mechanism has been evaluated by simulating a distributed policy based authorization and XACML access control system. A number of conflict policies of different nature have been injected in the policy database and conflicts have been identified through proposed XACML conflict policy detection algorithm. The implementation results show that the mechanism efficiently detects conflict policies having conflicts with respect to subject, resource and action attributes. This demonstrates that the approach is workable and can be used to detect conflict policies among a set of XACML policies.

KEYWORDS

XACML, Access Control Policies, Conflict Policies, Policy-based Authorization Framework

1. INTRODUCTION

A large scale distributed system is an interconnected set of heterogeneous autonomous systems that cooperatively solve a large problem. The problem is generally divided into a number of independent tasks that are executed in parallel or distributed over different nodes around the system for individual processing. These heterogeneous autonomous administrative domains, which are part of the distributed system, use and provide resources that can be shared among different members of the distributed system based on their authorization status and their conformance to established policies.

Distributed systems can be categorized as Distributed Computing Systems and Distributed Information System [1]. A Distributed Computing System is used for high performance computing. Here the emphasis is on integrating and exploiting compute power of different machines available over the system. A Distributed Information System is used to manage huge information / large databases with emphasis on interoperability. This class has its applicability

mainly in distributed databases [1]. Distributed Computing Systems can be further categorized into Cluster Computing Systems and Grid Computing Systems but other types like peer-to-peer computing, autonomic computing, pervasive computing, ubiquitous computing and internet computing etc., are also prevalent. The developments in the field of distributed computing have given rise to complex virtual systems. These systems enable sharing of resources and services distributed over geographically dispersed, heterogeneous, autonomous administrative domains and allow one to economically and efficiently perform a compute or storage intensive task by harnessing the services/features available over other domains in the system.

The resources and services available over a distributed system (e.g. grid or web) are generally protected by authorization and access control policies. The resources and services have complex access control policies associated with them and access is granted based on conformance to these policies [2]. Policies on a particular resource/service are stated by respective owners or policy administrators through policy specification tools/applications. The policies go on changing from time to time. One of the major problems that occur as a consequence of changing policies time to time is of introduction of conflicts among policies in a policy database. This may have serious consequences. The problem becomes more prevalent if there are no mechanisms/automated tools to identify and resolve conflict policies. This paper presents the design, implementation and evaluation of a mechanism that can be used by resource/service providers or policy administrators to proactively detect conflict XACML policies among a set of policies in a policy database.

The paper is organized as follows. Section II describes the background and related work in the field of access control policies, particularly focusing on XACML [3]. Section III presents the proposed XACML conflict policies detection mechanism. Implementation and evaluation details have been presented in section IV and finally, section V concludes the results with summary and future scope.

2. BACKGROUND AND RELATED WORK

A Policy in general terms can be defined as a definitive goal, course, or method of action based on a set of conditions to guide and determine present and future decisions [2]. Policies are implemented and utilized in a particular context. E.g. policies may include aspects of security, workload, networking services, business process and many other areas [2]. The policy can also be of a particular type e.g. authentication policy, trust policy, privacy policy and authorization policy etc [4]. Authentication policies deal with authentication related issues e.g. authentication mechanisms and type of security credentials required by subjects for the access of a particular resource/service. Privacy policies describe privacy related concerns among service providers and requesters e.g. access of personal information for a particular purpose by a particular subject. Trust policies describe trust related concerns among service providers and requesters e.g. accessing the service of a particular domain only if the trust with that domain is greater than a specified threshold value. The methods of policy specification, validation, evaluation, and enforcement etc. are different for policies of different types [4]. In this paper we are mainly concerned with access control policies which are expressed in XACML. XACML is an OASIS standard which can be used and extended to specify any general or specific access control policy in an interoperable way [3]. A XACML policy can be extended to include the aspects of authentication, privacy, trust, business processes, network workload etc.

A number of projects like Cardea [5], EGEE [6], Entrust [7], Globus [8], PERMIS [9], PRIMA [10], [11], [12] etc. are either using XACML to express access control policies or are influenced from XACML. Cardea [5] is a distributed system that enables dynamic access control and incorporates SAML and XACML standards. In Cardea, XACML plays a critical role within the distributed authorization framework [5]. In EGEE (Enabling Grids for E-sciencE) project,

XACML has been used to obtain authorization decisions [6]. It allows basic interoperability and pluggability of the authorization components from multiple authorization domains in OGSA framework [6]. It is also being used by Entrust [7] to capture fine-grained entitlements policies in a standardized and interoperable syntax. The authorization and access control framework of Globus Toolkit® 4, which supports multiple policies, also make use of XACML and enable multipolicy authorization. PERMIS [9] implements role based access controls using X.509 attribute certificates and shares similarities with XACML [3]. PRIMA [10] focuses on the issues of management and enforcement of fine-grained privileges. In encodes privileges in XACML. It uses X.509 attribute certificates to bind privilege attributes to subjects and policies to resources [10]. In PRIMA, resource and privilege management policies are created in XACML.

The XACML policies are written by policy administrators or resource/service owners either manually or using automated tools. If a resource/service has multiple owners and multiple policies associated with it, then it increases the probability that some of the policies will conflict with each other. The conflicts may be with respect to subjects, resources/services, actions or any other environment attributes. To indentify any type of conflict, it is very important and necessary to have a deep understanding of XACML Policy Model itself. Following section briefly describes the XACML Policy Language Model.

The major components of policy language model are i) Policy Set ii) Policy and iii) Rule. A policy set describes the set of policies and is a combination of target, policy combining algorithm, a set of policies and obligations. A Policy describes a set of rules and is a combination of target, rule combining algorithm, a set of rules and obligations. A rule is the most elementary unit of policy [3]. These are generally encapsulated in a policy. A rule is a combination of a target, an effect and a condition. Following table represents the major components of the XACML policy language model along with their brief description.

Table 1. XACML Components

XACML Component	**Meaning**
Policy Set	Describes the set of policies, other policy sets, policy combining algorithm, target and obligations. It governs access for a specific subject, resource or other components of target.
Policy	Describes the set of rules, rule combining algorithm, target and obligations. Policies generally consist of one or more rules.
Policy Combining Algorithm	Describes the procedure for combining results from multiple policies and obligations, i.e. how the results from multiple policies and obligations are to be interpreted in a combined form.
Obligations	Describes the set of operations to be performed by policy enforcement point along with enforcement of policy.
Rule	Describes the set of target, effect and condition and is generally a component of policy.
Condition	Describes an expression that generally evaluates to true, false or indeterminate.
Effect	Describes the outcome of a rule, generally in the form of ‘deny’ or ‘permit’.
Rule Combining Algorithm	Describes the procedure for combining results from multiple rules and obligations, i.e. how the results from multiple rules and obligations are to be interpreted in a combined form.
Target	Describes the set of subject, resource and action to which the rule, policy or policy set is applicable.

Subject	Describes an actor who initiates the access request or to whom the rule, policy or policy set is applicable.
Resource	Describes data, service or any other system component present in access request or to whom the rule, policy or policy set is applicable.
Action	Describes the operation to be performed on a resource.
Environment	Describes the set of attributes which are generally independent of a particular subject, resource or action.

Figure 1 shows the association of different components of XACML policy language model as described in [3].

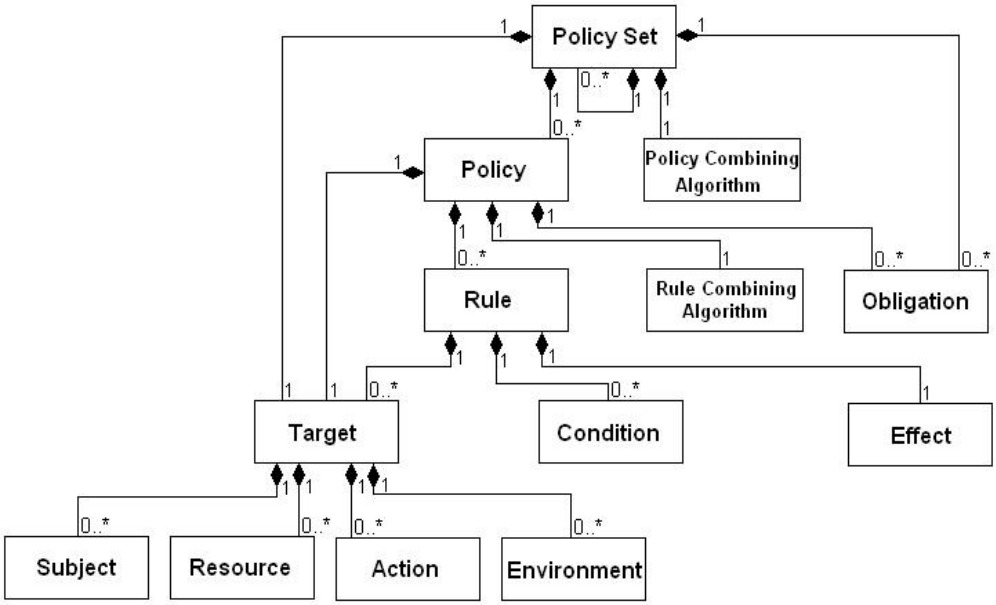

Figure 1. XACML Policy Language Model [4]

As shown in Figure 1, a policy set may have zero or more policy sets, zero or more policies, one policy combining algorithm and zero or more obligations. A policy may have zero or more rules, one rule combining algorithm, one target and zero or more obligations. A rule may have zero or more targets, zero or more conditions and one effect. A target may have zero or more subjects, zero or more resources, zero or more actions and zero or more environment attributes.

Figure 2 shows the skeleton of a XACML policy. In this figure, different components of XACML policy language model have been shown as either XML elements or XML attributes.

It can be easily interpreted from Figure 1 and Figure 2 that a XACML policy consists of a number of components and its XML version is very lengthy, as it involves a number of XML elements. In the absence of any automated XACML policy generation tool, it is very difficult to write and understand such policies [4]. Similarly it is very difficult to identify conflicts among a set of XACML policies. The conflict can be present with respect to subject, resource, action or any other attribute.

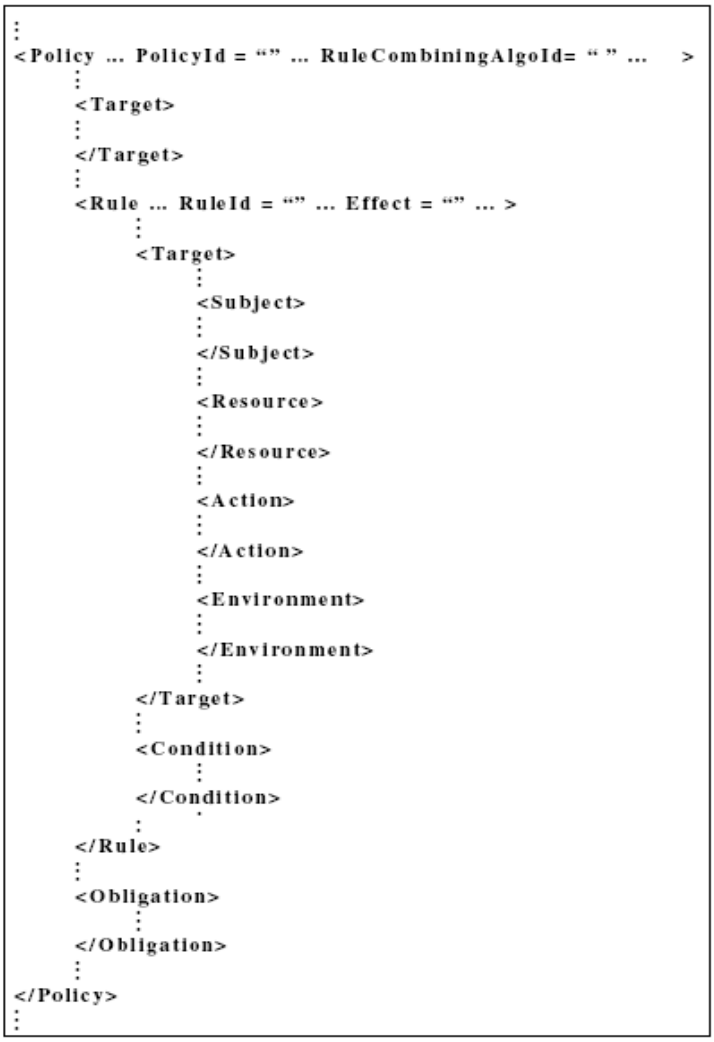

Figure 2. XACML Policy Skeleton [4]

In a large XACML policy database, there can be number of conflicts of different nature. A conflict may be valid or invalid. For example, a policy P1 may say that user U-1 can perform action A-1 on resource R-1 any time. Another policy P2 may say that Action A-1 cannot be performed on resource R-1 by any user on Wednesday. Clearly, these policies conflict with respect to user U-1 performing action A-1 on resource R-1 on Wednesday. Now human judgment is required to decide whether this situation be considered as a conflict or not. If this is not a conflict then either P1 is allowed to supersede P2 or P2 is allowed to supersede P1. But if this is a conflict, then either P1 or P2 needs to be altered. So it depends upon human interpretation that whether a policy is "conflict policy" or not. In this paper we have made an attempt to design and evaluate a mechanism to identify conflicts among a set of policies. The decision regarding whether the identified conflict is valid or invalid, has been left to the policy administrator. Following section describes the proposed conflict detection mechanism.

3. XACML CONFLICT POLICIES DETECTION MECHANISM

In order to identify conflicts with respect to subject, resource, action and other attributes and their combinations, it is necessary to scan each and every policy in the database and to compare

that policy with the rest of the policies to determine whether a conflict exists or not. This concept has been used to implement the conflict policy detection mechanism. Following figure describes this mechanism in pseudo code form.

```
pc: policy counter
cx: current xacml policy
cs: current subject
cr:current resource
ca: current action
i: loop counter
sx: stored xacml policy
ss: stored subject
sr:stored resource
sa: stored action
cpd: current policy decision
spd: stored policy decision
getXACMLPolicy(i):method that returns ith  xacml policy from policy database
getApplicableSubject(p): method that returns subjects which are applicable on policy p
getApplicableResource(p): method that returns resources which are applicable on policy p
getApplicableAction(p): method that returns actions which are applicable on policy p
getEvaluationDecision(p): method that returns the evaluation result of policy p
markConflict(x, y): method that marks conflict policies, x and y, i.e. it tags policies which conflict
with each other in the database
xacmlConflictDetection( )
{
 for (pc=1; pc<number of policies in xacml policy  database; pc=pc+1)
 {
            String cx=getXACMLPolicy(pc);
            String cs=getApplicableSubject(cx);
            String cr=getApplicableResource(cx);
            String ca=getApplicableAction(cx);
            Decision cpd=getEvaluationDecision(cx);
            if(cpd!= “indeterminate”)
            {
              for(i=pc+1;i< number of policies in xacml policy database; i=i+1)
              {
                        String sx=getXACMLPolicy(i);
                        String ss=getApplicableSubject(sx);
                        String sr=getApplicableResource(sx);
                        String sa=getApplicableAction(sx);
                        if(cs==ss and cr==sr and ca==sa)
                        {
                                  Decision spd=getEvaluationDecision(sx);
                                  if(spd!=“indeterminate”)
                                  {
                                            if(cpd!=spd)
                                            {
                                                      markConflict(pc, i);
                                            }
                                  }
                                  else
                                            continue;
                        }
            }
  }}}
```

Figure 3. Pseudocode for conflict policy detection mechanism

Figure 3 describes the logic of xacmlConflictDetection() method in pseudocode form that has been used to find conflicts among a set of policies in a policy database. The algorithm works by comparing the policies that have the same target (i.e. subject, resource and action elements) because only the compatible policies need to be evaluated and compared. The policies are evaluated and evaluation decision is returned by the method getEvaluationDecision(). If the evaluation result of any two compatible policies is different, then a conflict is noted and marked in the database using markConflict() method. If the evaluation result of any policy is "indeterminate" then the evaluation results need not be compared as it will not make any sense.

```
xp:xacml policy
sa: subject attributes
ra: resource attributes
aa: action attributes
getSubjectAttributes(s): method that returns attributes of subject s
getResourceAttributes(r): method that returns attributes of resource r
getActionAttributes(a): method that returns attributes of action a
getApplicableSubject(p): method that returns subjects which are applicable on policy p
getApplicableResource(p): method that returns resources which are applicable on policy p
getApplicableAction(p): method that returns actions which are applicable on policy p
getApplicableConditions(p): method that returns applicable conditions on policy p
getApplicableRules(p): method that returns applicable rules on policy p
getDecision (xp, sa, ra, aa, cd, rl): method that returns evaluation decision of a policy xp with
respect to subject attributes sa, resource attributes ra, action attributes aa, conditions cd and rules rl

getEvaluationDecision(String xp)
{
        SubjectAttributes sa=getSubjectAttributes(getApplicableSubject(xp))
        ResourceAttributes ra=getResourceAttributes(getApplicableResource(xp))
        ActionAttributes aa=getActionAttributes(getApplicableAction(xp))
        Rules rl=getApplicableRules(xp);
        Conditions cd=getApplicableConditions(xp);
        Decision decision=getDecision(xp, sa, ra, aa, cd, rl);
        return decision;
}
```

Figure 4. Pseudocode for method getEvaluationDecision

Figure 4 describes the getEvaluationDecision() method in pseudocode form. To evaluate a policy, the subject, resource and action attributes are fetched from XACML policy along with rules and conditions. The rules and conditions are evaluated by making use of fetched attributes and evaluation decision is prepared. The evaluation decision is then returned and compared by the xacmlConflictDetection() method with the evaluation result of other compatible policy to determine whether a conflict exists.

4. IMPLEMENTATION DETAILS AND RESULTS

The implementation of the proposed mechanism has been done on .NET platform with MS SQL Server as database for storing XACML policies. For evaluation of conflict policies detection mechanism, we have considered a set of 50 XACML policies. These policies have been created using custom built XACML policy specification tool. These policies refer to different subjects, resources and actions that can be performed on them. The policies, targets, rules and conditions have been expressed in proper XACML format using the policy specification tool. Different number of conflicts (ranging from 1 to 25) with respect to subjects, resources and actions have been identified. The identified conflicts have been injected into the set of XACML policies by altering policy, target, rules or condition tags of a policy.

The performance has been evaluated by noting the time taken by the algorithm to detect different conflicts. For evaluation purpose, 25 different types of conflicts (with respect to subjects, resources and actions) have been identified and injected. Following graph shows the time taken by the algorithm to detect individual conflict from a set of 50 policies.

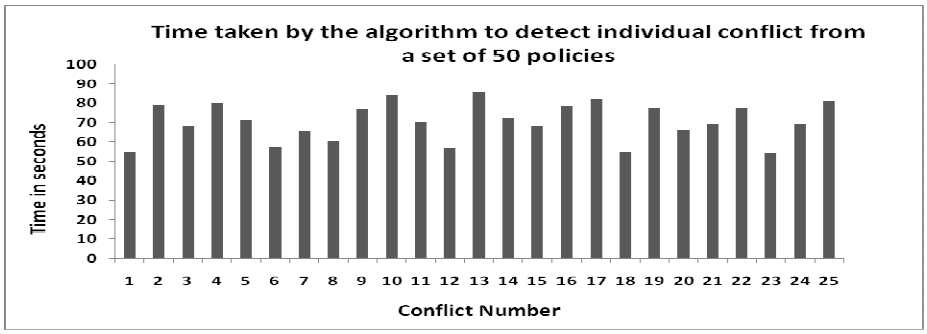

Figure 5. Time taken by the algorithm to detect individual conflict from a set of 50 policies

The average time to detect individual conflict from a set of 50 XACML policies has come out as 70.519 seconds. The experiment has also been performed by the keeping the conflicts fixed and increasing the number of policies in the policy database. The policies have been increased up to 200 in the ratio of 25. The average time to detect individual conflict from a set of 75, 100, 125, 150, 175 and 200 policies have been noted as 130.485, 237.214, 371.185, 539.822, 714.511 and 963.384 seconds. Figure 6 shows these details.

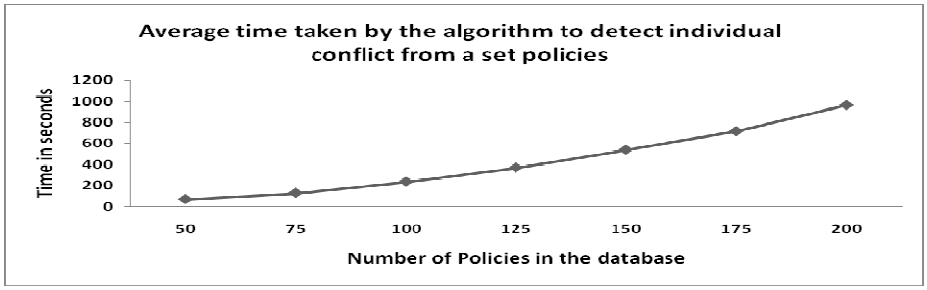

Figure 6. Average time taken by the algorithm to detect individual conflict from a set policies, having policies ranging from 75 to 200

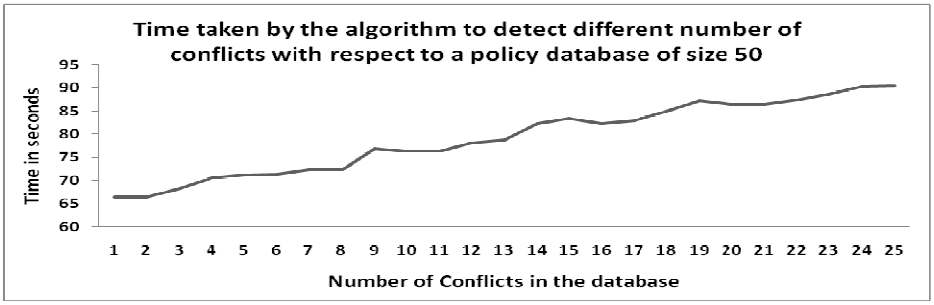

Figure 7. Time taken by the algorithm to detect different number of conflicts with respect to a policy database of size 50.

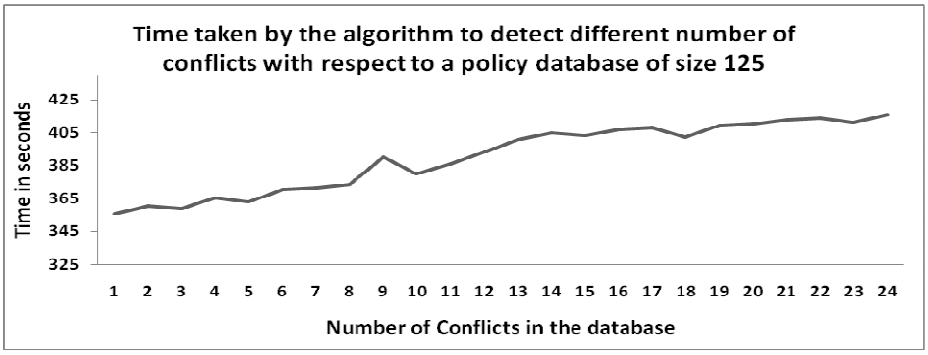

Figure 8. Time taken by the algorithm to detect different number of conflicts with respect to a policy database of size 125.

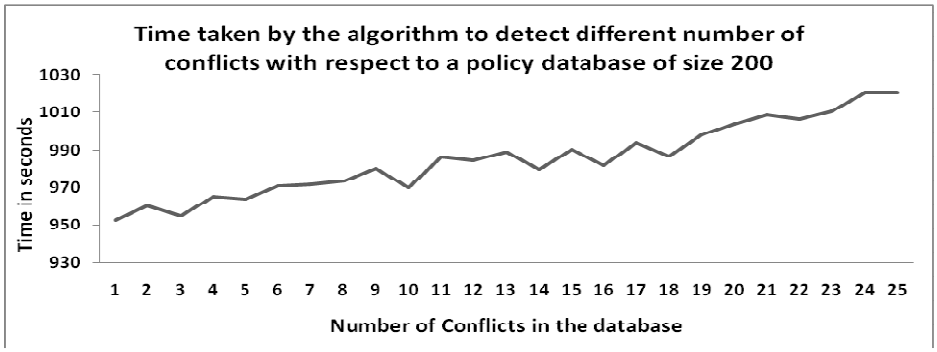

Figure 9. Time taken by the algorithm to detect different number of conflicts with respect to a policy database of size 200.

The experiment has also been performed by increasing the number of conflicts, one by one, in the database. In this case also the time taken by the algorithm to detect all conflicts applicable at a particular moment of time has been noted. Figure 7, Figure 8 and Figure 9 show the details for a set of 50, 125 and 200 XACML policies.

From the experimental results, it is clear that the time taken by the algorithm to detect conflicts increases linearly with the increase in number of conflicts and number of policies in the database. As time does not shoot up exponentially, the approach is workable.

5. SUMMARY CONCLUSION AND FUTURE SCOPE

The paper presents a mechanism to detect conflict XACML policies among a set of policies in a policy database. The performance analysis of the proposed algorithm has been done by i) noting the time taken by the algorithm to detect individual conflicts from a set of policies and ii) by injecting the conflicts one by one and noting the time taken by the algorithm to detect a set of conflicts from a set of policies. From experimental results it is clear that the time taken by the algorithm to detect conflicts increases linearly with the increase in number of conflicts and number of policies in the database. The time does not shoot up exponentially and hence the approach is workable. In future, we are planning to extend the algorithm to detect conflicts with respect to a combination of parameters i.e. the conflicts that involve a combination of target, subject, resource, action and other environment attributes.

REFERENCES

[1] A. S. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, Second Edition, Prentice Hall, October 2006.

[2] Joshy Joseph and Craig Fellenstein, "Grid Computing", Pearson Education, 2004.

[3] XACML Version 2, Available at http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[4] Sarbjeet Singh, "A Security Policy Framework for Grid Services", PhD Thesis, Computer Science & Engineering, Thapar University, Patiala, 2009.

[5] R. Lepro, "Cardea: Dynamic Access Control in Distributed Systems", NAS Technical Report NAS-03-020, November 2003.

[6] http://www.eu-egee.org/

[7] http://www.entrust.com/standards/xacml.htm

[8] http://www.globus.org

[9] D. W. Chadwick, A. Otenko and E. Ball, "Implementing Role Based Access Controls Using X.509 Attribute certificates - the PERMIS Privilege Management Infrastructure", 2002, available at http://sec.isi.salford.ac.uk/download/InternetComputingPaperv4.pdf

[10] M. Lorch, D.B. Adams, D. Kafura, M.S.R. Koneni, A. Rathi and S. Shah, "The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments", Fourth International Workshop on Grid Computing, Phoenix, Arizona, November 17, 2003, pp. 109.

[11] Han Tao, "A XACML-based Access Control Model for WebService", International Conference on Wireless Communications, Networking and Mobile Computing, 2005, pp. 1140-1144.

[12] J. Wu, C. B. Leangsuksum, V. Rampure and H. Ong, "Policy-based Access Control Framework for Grid Computing", Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), Singapore, May 16-19, 2006, pp. 391-394.Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC *Transactions on ECE*, Vol. 10, No. 5, pp120-122.

Authors

Kamalbir Singh received his B.Tech degree in Computer Science & Engineering from Punjab Technical University, Jallandhar, Punjab, India, in 2004. Currently he is pursuing M.E. degree in Computer Science & Engineering from Panjab University, Chandigarh, India. His research interests include distributed systems, distributed policy management and conflict policy identification and resolution mechanisms.

Sarbjeet Singh received his B.Tech degree in Computer Science & Engineering from Punjab Technical University, Punjab, India in 2001 and M.E. degree is Software Engineering from Thapar University, Patiala, India in 2003. He also received Ph.D degree in Computer Science & Engineering from Thapar University, Patiala, India in 2009, working on grid security systems architecture.
Currently he is working as Assistant Professor in Computer Science & Engineering at Panjab University, Chandigarh, India. He has more than 10 research publications in international conferences and journals to his credit. His research interests include parallel and distributed systems, distributed security architectures, distributed services like grid and web services, privacy and trust related issues in distributed environments.
Dr. Singh is a life member of Computer Society of India and Indian Society for Technical Education, India.