Published in final edited form as:
Proceedings (IEEE Int Conf Bioinformatics Biomed). 2012 January 2; 2011: 590–594. doi:10.1109/BIBM.2011.79.

Detection of Conflicts and Inconsistencies in Taxonomy-based Authorization Policies

Apurva Mohan,
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Douglas M. Blough,
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Tahsin Kurc,
Center for Comprehensive Informatics, Emory University, Atlanta, GA, USA

Andrew Post, and
Center for Comprehensive Informatics, Emory University, Atlanta, GA, USA

Joel Saltz
Center for Comprehensive Informatics, Emory University, Atlanta, GA, USA

Apurva Mohan: apurva@gatech.edu; Douglas M. Blough: doug.blough@ece.gatech.edu; Tahsin Kurc: tkurc@emory.edu; Andrew Post: arpost@emory.edu; Joel Saltz: jhsaltz@emory.edu

Abstract

The values of data elements stored in biomedical databases often draw from biomedical ontologies. Authorization rules can be defined on these ontologies to control access to sensitive and private data elements in such databases. Authorization rules may be specified by different authorities at different times for various purposes. Since such policy rules can conflict with each other, access to sensitive information may inadvertently be allowed. Another problem in biomedical data protection is inference attacks, in which a user who has legitimate access to some data elements is able to infer information related to other data elements. We propose and evaluate two strategies; one for detecting policy inconsistencies to avoid potential inference attacks and the other for detecting policy conflicts.

Keywords

Authorization policy; Biomedical ontology; Inference attacks; Policy conflicts

I. Introduction

Securing biomedical information presents challenges to information systems. An increasing number of biomedical databases make use of ontologies and semantic information; data that reside in such databases are mapped to domain ontologies. For instance, studies supported by the Atlanta Clinical and Translational Science Institute, which are the main motivating applications for our work, employ a variety of controlled terminologies including SNOMED-CT (Systematized Nomenclature of Medicine - Clinical Terms) concepts, LOINC (Logical Observation Identifiers Names and Codes) terms, and ICD9 (International Classification of Diseases) codes for the values of data elements. Queries against semantic

databases can return results based on not only the values of individual data elements, but also *explicit* and *inferred* relationships, such as class-subclass relationships, specified in ontologies. Moreover, biomedical databases can be accessed by a wide range of users and, in the case of large scale collaborative studies, across multiple institutions. This requirement dictates that multiple data access control policies be implemented and managed. Thus, the security infrastructure for large databases of biomedical data must (1) ensure that policies do not conflict with each other, new policies or changes to policies do not create conflicts that may allow unauthorized access to data, and (2) take into account semantic information in biomedical databases, support policies defined on concepts from ontologies, and be able to detect conflicts in such policies.

In this work, we present strategies for conflict detection when access control policies are defined on taxonomies, which represent hierarchical relationships (i.e., class-subclass relationships) between concepts. Access control rules can be defined for any of the concepts in the taxonomy. That is, a rule specifies whether a request on data elements, whose values are mapped to the corresponding concept, should be granted or denied. Access control rules for a concept can differ from its children and they can be in conflict. Detecting and resolving these conflicts is non-trivial because it involves identification of applicable rules and detecting conflicts among them dynamically during execution of data access requests. We propose a dynamic conflict detection and resolution strategy and we have developed an efficient algorithm to carry out this strategy. Our work is also concerned with attacks where a principal who has legitimate access to some node is able to infer data related to another node. Our approach to prevent this type of inadvertent data disclosure is by ensuring policy consistency, meaning that the framework ensures that a node which can lead to inference about other nodes is protected by the same level of authorization policies as the other nodes. We have developed an algorithm to check policy consistency to detect potential information inference vulnerabilities. The execution times of these two algorithms are evaluated empirically.

Prior work most closely related to our research is the work by Jajodia, et al. [3], which proposes some strategies for conflict resolution and authorization propagation. However, their work does not consider inference relations nor dynamic analysis of conflicts. Inference attacks are common against statistical databases [4], [7], where the results of repeated statistical queries are used to try to infer specific database elements. Some approaches used to protect privacy against these attacks are: k-anonymity [11], l-diversity [6], and t-closeness [5]. Here, we consider inference attacks where some specific elements of the database (not statistical results) can be used to infer other database values. Researchers have proposed various methods to reduce an attacker's access to data that is highly correlated to sensitive protected data, e.g. [2]. These methods are not designed against attackers who have legitimate access to some information that is highly correlated to the non-accessible protected information. Several works have considered how to represent inference relations between different data items [14], but are not focused on detection. Thuraisingham et al. developed methods for detecting inference relations during database design time and presenting them to the system administrator [12]. In contrast to [12], our approach addresses the problem using consistent policies and is not concerned with database design. Several projects have developed support for detecting inferences related to duplicated data protected using inconsistent policies [10], [13]. These approaches consider only data elements that are exact copies of each other.

II. Problem Description

A taxonomy can be represented as a tree, in which different levels in the tree correspond to class-subclass hierarchies. For example, if 'flu' is a class (or concept), then the specific

types of ‘flu’ will be its sub-classes and will be represented as child nodes in the hierarchy. Our approach views a biomedical ontology as a *resource tree*, as is illustrated in Figure 1. Each node in the tree corresponds to an ontology concept and represents *a resource to be protected*. Access control policies defined on a given node (concept) specify whether access requests to data elements, whose values are mapped to the corresponding concept, are to be granted or denied and under what conditions. In the figure, some tree nodes, such as n_1, n_2, n_8, n_9, and n_{19}, have an access control rule displayed next to them. The effect of a rule is denoted by the letter ‘P’ or ‘D’ representing a response of ‘Permit’ or ‘Deny’ – that is, the request is either *Permitted* or *Denied*. The elements on the left denote the subject, environmental, and action attributes represented by ‘s’, ‘e’ and ‘a’.

Authorization flows are always from any node towards its children nodes as shown in Figure 1. That is, an access policy defined on a node should be enforced for all its children as well. For instance, assume ’flu’ has two subclasses; ’common flu’ and ’swine flu’. If a ’Deny’ policy is defined on ’flu’ – any request to retrieve data mapped to ’flu’ is denied –, all requests to the subclasses of ’flu’ should also be denied.

Different authorization rules can be specified on different levels of ontologies. Each node either has its own access control rules or inherits them from its parent. Conflicts and inconsistencies may be introduced because different granularity of data are protected by different policy rules. In case a node’s access control rules conflict with its parent, conflict resolution should be performed. It is required that policies on all ontological classifications (nodes on the resource tree) in a database are synchronized. This is required to ensure consistency in authorization decisions on multiple paths leading to the same resource. If this is not done, then a user may be permitted to access a resource if he selects one access path, while he may be denied access through another path.

In addition to conflicts, inconsistencies may arise when there is an inference relationship between classifications on ontologies (nodes on a resource tree). In databases inference attacks are used to infer sensitive information which a subject does not have access to by using sensitive or non-sensitive information that he has access to. *Inference relations* between different nodes exist when one or more nodes can be inferred from some related node. We treat this condition as a policy inconsistency and refer to such a condition as *inference inconsistency*. In the current research we assume that inference relations between concepts in an ontology are pre-determined and provided to our conflict and inconsistency detection algorithms as input.

To illustrate inference inconsistency, let us consider a patient who is diagnosed as HIV positive. The patient’s documents are stored in a database where the data elements are mapped to a medical ontology. Since the patient’s condition is highly sensitive, access to his diagnosis information should only be provided to individuals with appropriate authorization such as the patient’s doctor. Assume that a node L1 in the ontology maps to the HIV status data elements. This node must be protected by a policy rule that denies access by any user but the patient’s doctor, Dr. Brown:

1. {(Dr. Brown), (Node L1), (Read, Write)} = Permit
2. {(*), (Node L1), (*)} = Deny

The second rule uses the wildcard * which represents all the users other than the ones in the adjoining rule (Rule 1). The second rule denies any other user access to node L1. However, it is possible to infer a patient’s HIV status from his/her laboratory tests. Consider nodes L3, L4 and L5 which correspond to different types of blood tests. The patient’s document in these categories may have highly sensitive information which may lead to an inference

about the patient's diagnosis. The following policy rules allow any researcher and volunteer nurse to access these nodes:

1. {(Any researcher), (Node L3, Node L4, Node L5), (Read, Write)} = Permit
2. {(Any volunteer nurse), (Node L3, Node L4, Node L5), (Read)} = Permit

In this case, these rules create an inference inconsistency, since while the authorization system limits access to node L1, the rules allow access to nodes L3, L4, and L5, thus enabling an unauthorized person to deduce the patient's disease status. The authorization system should implement mechanisms to detect such inconsistencies to properly protect information.

III. Conflict and Inconsistency Detection Methods

Our work handles two problems - i) Authorization policy conflicts among different hierarchical levels in a resource tree; and ii) detection of inconsistencies in authorization policies specified for *inference related* nodes. We address the former by resolving the conflicts and inform these inconsistencies to the system administrator. In the latter case, *inference inconsistencies* can be by mistake or by design (e.g., as a result of a data collection and analysis workflow). As such our approach does not resolve them but provides a mechanism to detect and inform a system administrator of the inconsistencies. The inconsistencies can be classified into strong and weak inconsistencies and reported accordingly so that the database administrator can handle them according to their priority.

Policy analysis can be done in a static or dynamic manner. In the static analysis mode, all the authorization rules for a particular node will be compared with authorization rules for nodes above and below it to determine if there are some conflicts. In the dynamic analysis mode, on the other hand, policy checks are performed in real time when an access request is received by the system. When an access request for a node is received, the authorization system detects and resolves conflicts with the parents and children of the requested node and determines the data subset which the requesting user can access. Dynamic analysis is faster, but analyzes only the rules that are applicable to the current request and determines conflicts in them. Dynamic analysis also considers all the nodes above and below in hierarchy compared to the current node, but only considers the specific combinations of subject and environmental attributes present in the current access request.

We have employed the dynamic analysis mode, because conflict resolution and inconsistency detection can be done quickly (as is also shown in the experimental evaluation in Section IV). Our approach allows users to access only those data sets which they need in order to complete their tasks, while enforcing access control. In Figure 1, for example, if a user needs data sets n_{18} and n_{19}, then the user will get access to n_{12}, which contains only these data sets.

A. Conflict Handling Algorithm

We now describe the dynamic conflict analysis and handling algorithm and illustrate how it works with an example. The conflict handling algorithm is presented in greater detail in an accompanying technical report [8].

The user requests all the data associated with a node n. We create several arrays to help with data processing. Array1 contains the nodes at a level for which we have to evaluate policy rules; Array2 contains nodes from Array1 which allow data access for the user; *ReportArray* contains the children of node n which have a conflict with the policy specified for node n; and *FinalArray* contains the leaf nodes which are the children of node n children and are accessible by the user. The algorithm performs the following main steps:

1. An access request for node *n* is received with specified subject, environment, and action attributes.
2. The request is evaluated for each parent of node *n*. If the response is 'Deny' for any one of the parent nodes, the final response is set to 'Deny'. That is, the access request is denied.[1]
3. Node *n* is stored in Array2.
4. The request is evaluated for all the children of all nodes in Array2. The child nodes which have access decision different from node *n* are stored in the *ReportArray*.
5. The child nodes, which have a response of 'Permit' associated with them and are leaf nodes, are stored in the *FinalArray*.[2]
6. In the previous step, non-leaf child nodes with access decision 'Permit' are stored in an intermediate array and the ones with 'Deny' are neglected. Others are stored in Array2 which contains nodes whose children will be evaluated in the next iteration.[3]
7. We repeat steps 4 to 6 above till Array2 is empty.
8. The *ReportArray* is sent to the system administrator. The data elements based on the concepts (nodes) in the *FinalArray* are returned as the response to the user.[4]

B. Inference Inconsistency Detection Algorithm

The inference inconsistencies detection algorithm is presented in greater detail in an accompanying technical report [8]. The algorithm performs the following steps:

1. Access request for the node *n* is received with specified subject, environment, and action attributes.
2. The algorithm retrieves the list of all the nodes $n_{inferencenodelist}$ which can be inferred from the requested node *n*.
3. The access policy for node *n* is evaluated.
4. Access policy for each node in $n_{inferencenodelist}$ is evaluated and the ones which have access policy responses different from that of node *n* are reported to the system administrator.

Inference inconsistencies in the system can be a result of a mistake or can be included by design (e.g., as a result of a data collection and analysis workflow). Differentiating between these two is a reasoning problem and as such our approach does not resolve them but provides a mechanism to detect them and inform a system administrator of the inconsistencies. We assume that the system administrator will distinguish between the two and manually resolve them.

[1] An explicit 'Deny' rule is set on a node to prevent holders of those attributes from accessing any data on or below that node.
[2] If the decision on a node is 'Permit' or 'NotApplicable', then the more specific rule on the child node overrides. Since the actual data is only held on the lead nodes, we need to find all the lead nodes which are children of node n and see if the current requester is permitted or denied access to data on that lead node.
[3] This is in congruence with the two steps above. If there is an explicit 'Deny' on a node, the requester is prohibited from accessing data form any of the children of that node. On the other hand, if the decision is either 'permit' or 'NotApplicable', we continue to search for rules on child nodes which would override them.
[4] The *ReportArray* contains the nodes which have conflicting permissions than node n. The *FinalArray* contains leaf nodes whose data can be accessed by the requester.

IV. Experimental Evaluation

We have implemented the algorithms described in the previous section in Java programming language and used the XACML policy language and Sun's open source XACML engine [1] for specifying authorization policies. We have implemented some enhancements for performance improvement and to support the conflict detection and resolutions algorithms, in the policy component of the open source XACML engine. The experimental evaluation is targeted at examining the execution time of the conflict and inconsistency detection algorithms. We executed the experiments on a Linux server running Ubuntu 4.4.1. The hardware platform has 8 GB of RAM and Intel quad core Xeon(R) CPU 5150, 2.66GHz processor with 4096KB cache on each processor. We used the ontology provided in the i2b2 system [9]. We created synthetic policies for evaluation of performance of the algorithms in a controlled way.

A. Execution Time to Detect and Resolve Conflicts

This set of experiments investigates how long it takes to execute checks for policy conflicts in a hierarchical ontology. We have created a sample ontology from the ontology provided by i2b2 with a total of 10200 nodes (concepts). In our test setup, the root node represents the entire ontology and has three levels below it, where level 1 nodes are the direct children of the root node, and level 3 nodes are the leaf nodes.

We have measured system performance at three points while scaling down the number of nodes at 10,000, 6,000, and 2,000 nodes. For each level of the tree, a node is selected at random and an access request is generated to access the resource represented by that node. Permission to access data elements is evaluated according to the conflict detection algorithm. The performance results for this test case are presented in Figures 2 and 3.

For these experiments, we created policies by randomly selecting 10% of the total nodes and setting a rule with 'Permit' decision for a specified combination of subject, environment and action attributes. We repeated this procedure by setting 10% of the rules with 'Deny' rules.

Figure 2 shows the elapsed time to find all the authorized nodes for an access request for 10,000, 6,000, and 2,000 nodes for each level in the resource tree hierarchy with about 10% of the total nodes containing 'permit' authorization rules[5]. According to the conflict detection algorithm, if a parent node of the requested node has an explicit deny rule on it, then the child nodes are not searched and the request is denied. We show the total elapsed time as the sum of time for searching deny rules on parent nodes and time to search child nodes, which is the total elapsed time. We observe that if the access query is sufficiently narrowed down such that the resource is at level 2 or 3, then the time required for conflict detection and resolution is under one second. Figure 3 shows the elapsed time with the same parameters as Figure 2, but in this case the 10% authorization rules have the effect 'deny'[6].

A key observation from the results is that the total conflict handling time for a node is directly proportional to the number of its children in the subtrees.

B. Execution Time to Perform Consistency Check

In these experiments, we evaluate the execution time of consistency checks on policies involving nodes with inference relations. For example, if the requester requested to access node A and it has inference relationships with nodes B, C, and D s.t. $(B; C; D) \rightarrow A$, then

[5]If this authorization rule is matched, then the final effect is 'Permit'
[6]If this authorization rule is matched, then the final effect is 'Deny'

we have to make sure that node *A*, *B*, *C*, and *D* have the same level of protection. This check is required so that a user, who can access node *A* with a lower level authorization policies would not inadvertently learn about nodes *B*, *C* and *D* which are protected by more stringent policies.

In the evaluation, we generate a set of nodes which have an inference relationship with other nodes. As in the previous example, nodes *B*, *C* or *D* can be inferred by node *A*, i.e., (*B; C; D*) → *A*. Other nodes *E*, *F*, *G* lead a user to infer node *A*, i.e. *A* → (*E; F; G*). When an access request for node *A* is received by the authorization system, it checks whether the access control rules for node *A* are consistent with nodes *B*, *C*, and *D*. On the other hand, if an access request is received from nodes *E*, *F*, or *G* then it checks whether their access control rules are consistent with node *A*.

Typically the number of data classifications that can be inferred from another classification is relatively low, so we choose the number of related classifications accordingly. We assume that each randomly selected node can lead to an inference of about 3, 5, 10 and 20 nodes respectively and evaluate the time required to check the policy consistency for all these nodes by evaluating the algorithm presented in III-B. In Figure 4, we see that up to 10 nodes, the time to check policy consistency is under one second.

V. Conclusions

Security is a critical component in making biomedical data available for research purposes. Because of complexity of semantic databases and the variation of user access privileges, conflicts and inconsistencies may arise in a group of access control policies defined on ontology concepts for a database. Our work has investigated two algorithms designed to support (1) detection and handling of conflicting policies defined at different levels of a hierarchical ontology and (2) detecting and reporting *policy inconsistencies* among policies defined on inference related concepts in an ontology. These algorithms provide a tool for the database administrators to better protect sensitive and private data. Our empirical evaluation of the execution times of these algorithms indicates that the algorithms are fast and likely to incur little overhead to the overall execution of the security infrastructure and database system. Our current work assumes the inference relationships among concepts in an ontology are provided explicitly. We plan to investigate methods to detect such inference relationships in a future work.

Acknowledgments

This research is supported in part by the PHS Grant UL1RR025008 from the CTSA program, by the National Science Foundation under Grant CNS-CT-0716252, by R24HL085343 from the NHLBI, by Grant R01LM009239 from the NLM, and by SAIC/NCI Contract No. HHSN261200800001E from the National Cancer Institute, NCI Contract No. N01-CO-12400, 85983CBS43, and 94995NBS23.

References

1. Sun XACML Policy Engine. sunxacml.sourceforge.net
2. Chang, L.; Moskowitz, I. CIMCA01. 2001. A bayesian network schema for lessening database inference.
3. Jajodia S, Samarati P, Sapino ML, Subrahmanian VS. Flexible support for multiple access control policies. ACM TDS. Jun.2001
4. Jonge WD. Compromising statistical databases responding to queries about means. ACM TDS. 1983
5. Li N, Li T. t-closeness: Privacy beyond k-anonymity and l-diversity. IEEE Conf on Data Engineering. 2007

6. Machanavajjhala A, Gehrke J, Kifer D. l-diversity: Privacy beyond k-anonymity. ACM TKDD. Mar.2007
7. Mandujano S. Inference attacks to statistical databases: Data suppression, concealing controls and other security trends. Aleph Zero online magazine. Apr-May;2000
8. Mohan, A.; Blough, D.; Kurc, T.; Post, A.; Saltz, J. CERCS Tech Report GIT-CERCS-11-05. Georgia Tech; 2011. Detection of conflicts and inconsistencies in taxonomy-based authorization policies.
9. Murphy S, Weber G, Mendis M, Chueh H, Churchill S, Glaser J, Kohane I. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). Journal of AMIA. Mar-Apr;2010
10. Stoica A, Farkas C. Ontology guided xml security engine. Journal of IIS. 2004
11. Sweeney L. k-anonymity: A model for protecting privacy. Int Journal on UFKS. 2002
12. Thuraisingham B. The use of conceptual structures for handling the inference problem. Database Security V. 1992
13. Yang L, Junmo X, Jing L. Protecting xml databases against ontology-based inference attack. CIS. 2007
14. Yip R, Levitt K. Data level inference detection in database systems. IEEE CSFW. 1998

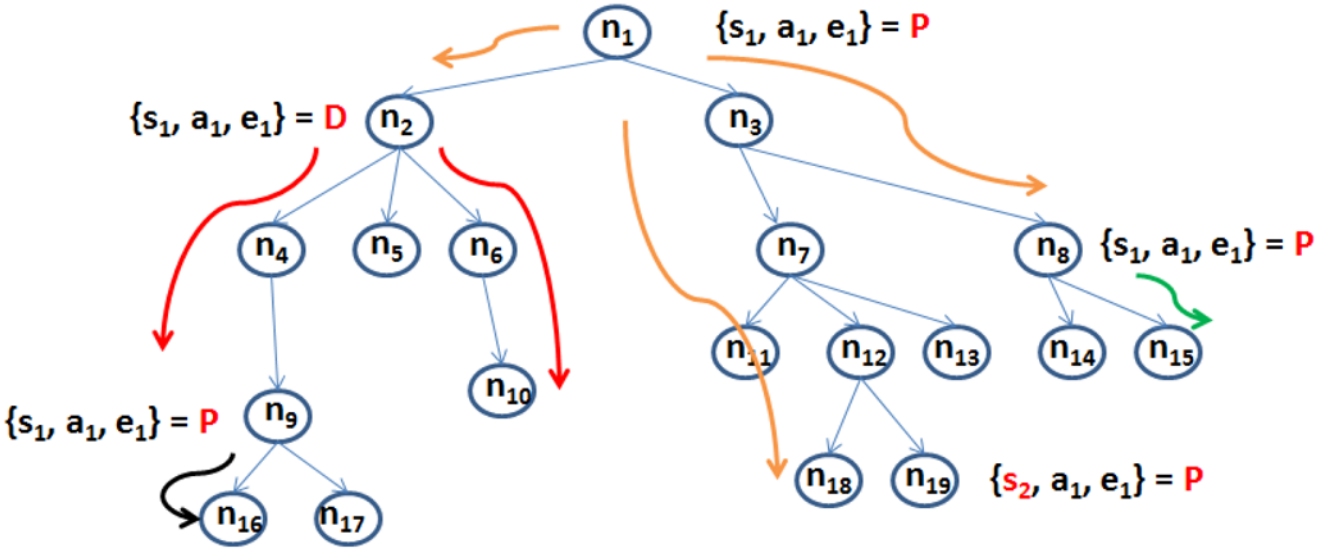

Figure 1.
An example tree with data element hierarchy.

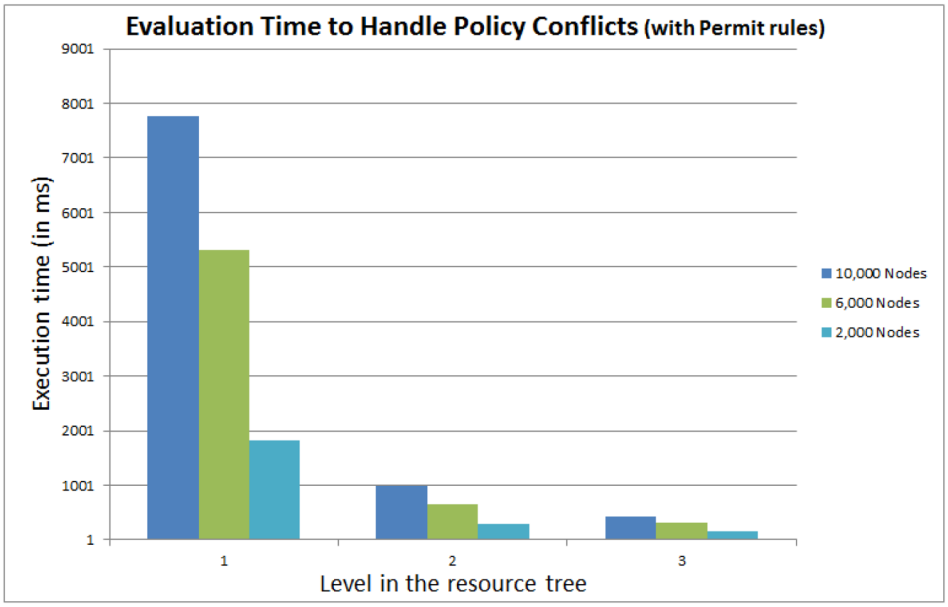

Figure 2.
Evaluation time to detect conflicts with permit rules.

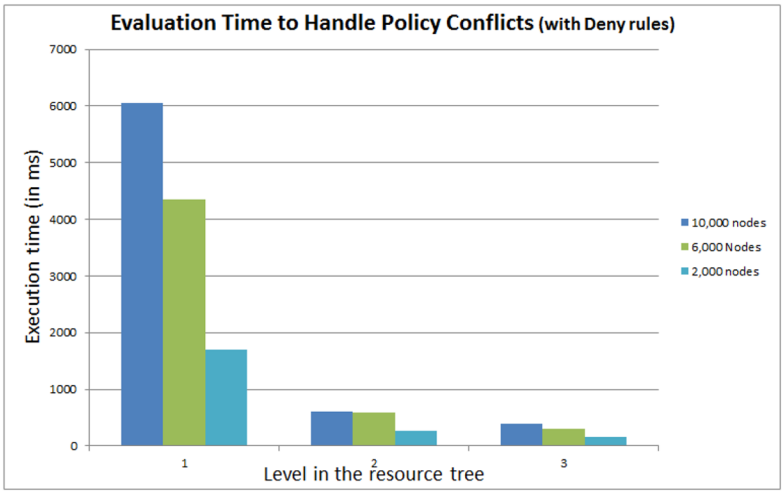

Figure 3.
Evaluation time to detect conflicts with deny rules.

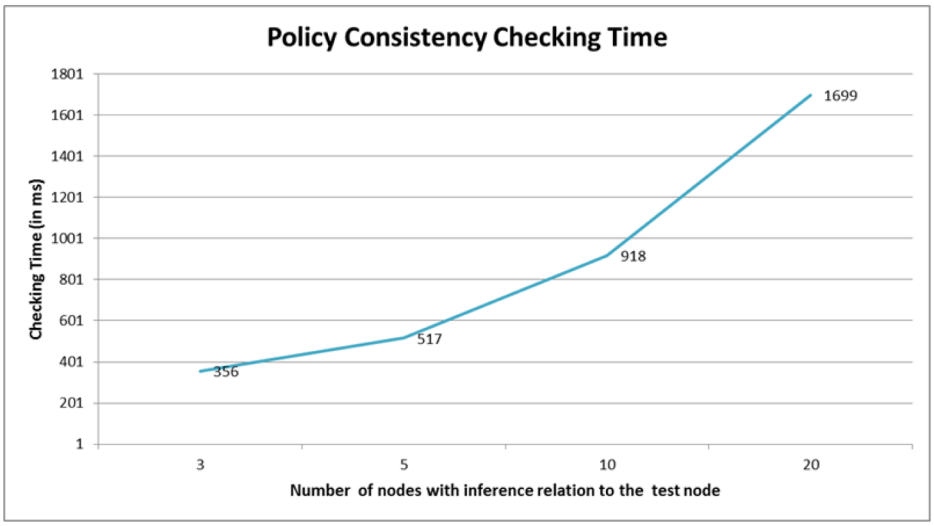

Figure 4.
Policy consistency checking time.