Detection of Conflicts and Inconsistencies in Taxonomy-based Authorization Policies

Apurva Mohan, Douglas M. Blough
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA
Email: apurva@gatech.edu, doug.blough@ece.gatech.edu

Tahsin Kurc, Andrew Post, Joel Saltz
Center for Comprehensive Informatics
Emory University
Atlanta, GA, USA
Email: {tkurc,arpost,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, and as such policy rules may conflict with each other, inadvertently allowing access to sensitive information. Detecting policy conflicts is non-trivial because it involves identification of applicable rules and detecting conflicts among them dynamically during execution of data access requests. It also requires dynamically verifying conformance with required policies and logging relevant information about decisions for audit. 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. This type of inadvertent data disclosure should be prevented by ensuring policy consistency; that is, data elements which can lead to inference about other data elements should be protected by the same level of authorization policies as the other data elements. We propose two strategies; one for detecting policy consistencies to avoid potential inference attacks and the other for detecting policy conflicts. We have implemented these algorithms in Java language and evaluated their execution times experimentally.**

***Keywords*-Authorization policy, Biomedical ontology, Inference attacks, Policy conflicts.**

I. Introduction

Translational and clinical research projects often require the collection, exchange, storage, and analysis of many types of data, including imaging, "omics", observational, and electronic health record data. Because of regulatory requirements and intellectual property concerns, sensitive and private datasets need to be protected by appropriate access control policies and mechanisms so that users can only access subsets of data that they are authorized to access. Without proper access control policies and software systems to enforce them, effective use of biomedical data in research, especially in collaborative research, and health care delivery can be hindered.

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. For instance, in the Atlanta Clinical and Translational Science Institute, biomedical data collected in research studies may need to be accessed by clinicians, investigators, postdoctoral researchers, biostatisticians with different access privileges and from partner 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 attacks. The execution times of these two algorithms are evaluated empirically.

The rest of the paper is organized as follows - Section II describes the problem with current ontology-based authorization systems. Section III describes the proposed solutions. Section IV presents an experimental evaluation of performance of our algorithms. SectionV discussed related work and finally Section VI concludes the paper.

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'. The subject (user) attribute represents a set of attributes characterizing the user (e.g., an investigator or a research nurse) requesting access; the action attribute defines the possible actions (e.g., read data, write data) that can be issued by the user and evaluated by this policy rule, and the environmental attribute represent the environmental characteristics (e.g., from which machines authorized access requests can be issued) associated with access requests.

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. For example, if nodes A and B convey sufficient information about each other, a user who has access to one of them can infer the other with high probability or even completely. In such a case, if A and B are protected by different policies where some users are authorized to access only A but can infer B form it. 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. We plan to investigate in a future work approaches for finding inference relations between concepts.

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. This ontology is used to specify access control on the data elements in the database. 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: *Ontology* $\rightarrow$ *Diagnoses* $\rightarrow$ *Infectious and parasitic diseases* $\rightarrow$ *Human immunodeficiency virus [HIV] disease*. 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

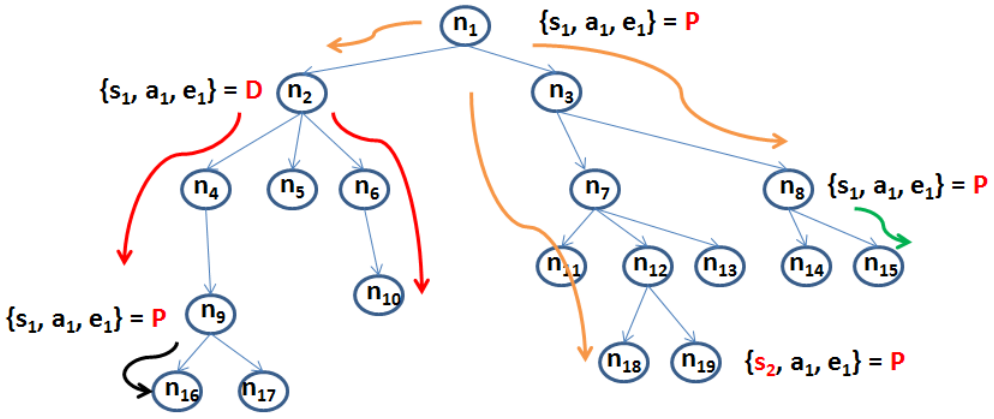

Figure 1. An example tree with data element hierarchy.

L3 (*Ontology* → *Lab Tests* → *Hematology* → *Complete Blood Count*), L4 (*Ontology* → *Lab Tests* → *Hematology* → *Blood Differential %* → *WBC Comments*), L5 (*Ontology* → *Lab Tests* → *Hematology* → *Blood Differential %* → *Total Cells Counted*). These nodes 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. There are two ways that conflicts can be handled. First, when policy rules are being composed, new rules are checked against the existing rules and an error is reported when a conflict is detected. In this way, the system does not allow the composition of conflicting policy rules. Second, the system allows the composition of conflicting rules but detects and resolves them before any access request is received. Since the number of combinations can potentially become very large, static analysis is generally much slower but more comprehensive.

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.

In our current work, we develop two novel algorithms for conflict handling and 'inference inconsistency' detection in 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 supports the principle of the least privilege through a fine-grained and inference consistency authorization model. That is, we allow 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 Figure 2. The algorithm performs the following steps:

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.

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' (Steps 5-9 in Figure 2). 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* (Steps 14-15 in Figure 2).
5) The child nodes, which have a response of 'Permit' associated with them and are leaf nodes, are stored in the *FinalArray* (Steps 16-17 in Figure 2).[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 (Steps 18-19 in Figure 2).[3]
7) We repeat steps 4 to 6 above till Array2 is empty (Steps 23-26 in Figure 2).
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 (Steps 27-30 in Figure 2).[4]

```
s, e and a represent the subject, environment and action attributes
R represents the request and Evaluate evaluates the access request

1       Receive request R(s,e,a) for node n
2       Create Array1, Array2, FinalArray, ReportArray, to hold node list
3       Variables temp, temp1 to hold a node

4       temp = n;
5       While(temp.parentid != NULL)
6               Evaluate[R(s,e,a), temp]   ->  Result
7               if(Result == 'Deny')
8                       return 'Deny'
9                       EXIT;

10      n.children -> Array1
11      while(Array1.hasmorelements)
12                      Array1.nextelement -> temp1
13                      Evaluate[R(s,e,a), temp1]   ->  tempResult
14                      if(tempResult != Result)
15                              temp1 -> ReportArray
16                      if(tempResult == 'Permit' && temp1 == leafnode)
17                              temp1 ->  FinalArray
18                      if(tempResult != 'Deny' and temp1 != leafnode)
19                              temp1 -> Array2

20      Array1.clear
21      if(Array2.size == 0)
22              GOTO STEP 27

23      while(Array2.hasmoreelements)
24              Array2.nextelement.children -> Array1.add
25              DELETE Array2.nextelement

26      REPEAT STEPS 11 TO 25

27      if(ReportArray.size > 0)
28              SEND[ReportArray, node n] to admin

29      return FinalArray
30      EXIT;
```

Figure 2. Conflict handling algorithm.

Consider the ontology shown in Figure 3. Different nodes have authorization rules as shown in Figure 4. S1, S2, S3 and S4 represent specific subject and environment attribute combinations and n1, ... , n8 represent the nodes in the resource tree. P and D represent the effects 'Permit' and 'Deny' respectively. Consider some example requests below:

1) If a request to access node n3 and its children from S2 is received, the algorithm will determine that there are no explicit 'Deny' rules on n3's parent nodes (i.e. n2 and n1). It will then evaluate authorization rules on n3's children. Since n4 has a 'Permit' for S2, only that leaf node will be returned.
2) If a request to access node n2 and its children from S2 is received, the algorithm checks that there is no

[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.

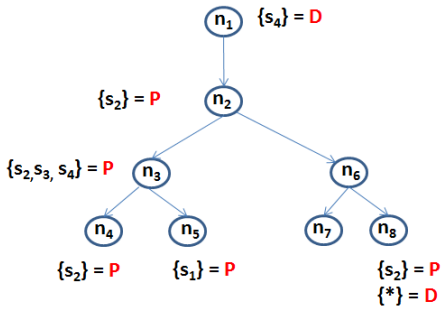

Figure 3. A sample ontology with authorization rules.

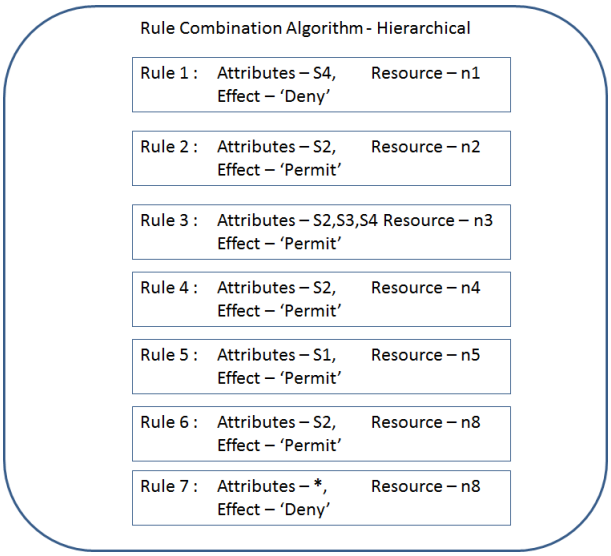

Figure 4. Authorization rules defined on the resource tree in Figure 3.

explicit 'Deny' on n1. Node n2's indirect descendants, n4 and n8, are the only once with 'Permit' rule for S2 and hence they will be returned. Since the decision on node n2 is 'Permit', nodes n6, n5 and n7 will be in the *ReportArray*.

3) If a request to access node n3 and its children with S4 is received, the algorithm will check for an explicit 'Deny' on n3's parents. Node n1 has an explicit 'Deny' for S4 and hence the request will be denied. The rule on n1 is read as *Any requester who holds subject attribute combination S4 is denied access to data classified under n1 and any of its children nodes.*

B. Inference Inconsistency Detection Algorithm

The inference inconsistencies detection algorithm is shown in Figure 5. 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.

```
s, e and a represent the subject, environment and action attributes
R represents the request and Evaluate evaluates the access request

1       Receive request R(s,e,a) for node n
2       Create Array1, ReportArray to hold node list
3       Variable temp to hold a node

4       n.Inferencenodelist -> Array1
5       Evaluate[R(s,e,a), node n]   ->  Result
6       While(Array1.hasnextelement)
7               A.nextelement -> temp
8               Evaluate[R(s,e,a), node temp]   ->  tempResult
9               if(tempResult != Result)
10              temp -> ReportArray
11      Send [ReportArray, node n] to admin
12      EXIT;
```

Figure 5. Inference inconsistencies detection algorithm.

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.

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 [3] for specifying authorization policies. We use XACML's standard request protocol, but the response protocol is modified so that the response contains a 'Permit' and also a list of all the data elements which are permitted to be accessed by the requesting user. We have implemented some enhancements, in terms of looping access requests, caching responses, using hierarchical relationships, implementing 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 [14] [2]. 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 6, 7 and 8.

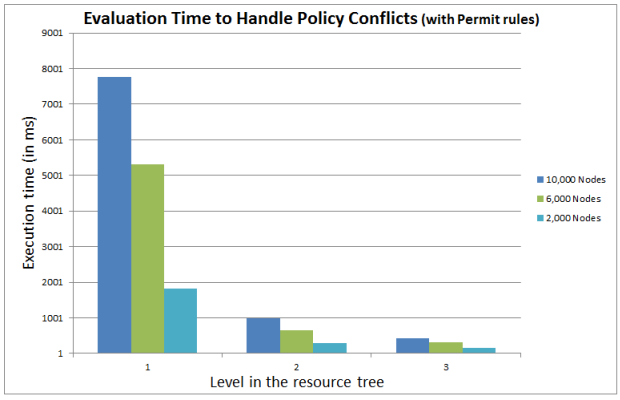

Figure 6. Evaluation time to detect conflicts with permit rules.

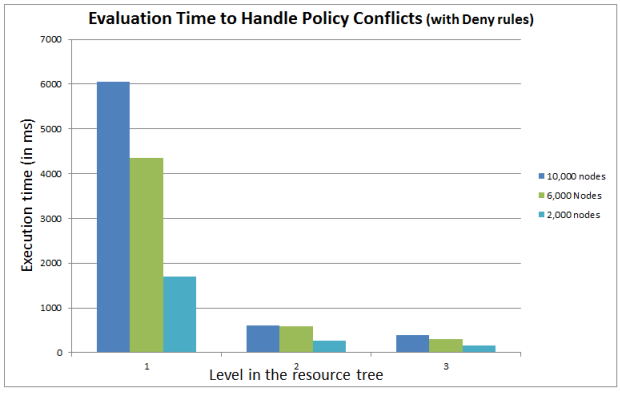

Figure 7. Evaluation time to detect conflicts with deny rules.

For these experiments, we created policies by randomly selecting 10% of the total nodes and setting a rule with

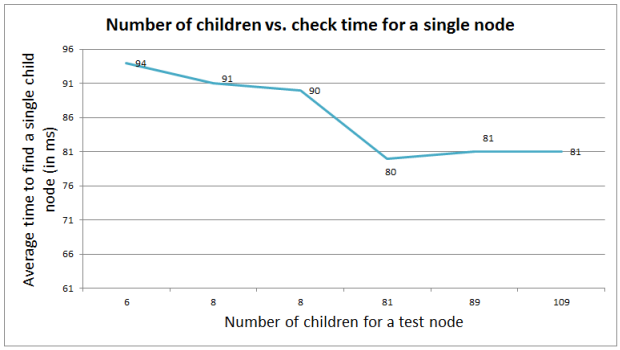

Figure 8. Average time to find a single child node and resolve its conflict.

'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 6 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 7 shows the elapsed time with the same parameters as Figure 6, 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. This result can be attributed to the fact that the nodes closer to the root node will have more children and will spend more time in detecting and resolving the conflicts. The average time to search a single child node at any level of hierarchy is almost constant (with a range of 80-94 ms) and is plotted in Figure 8. This time can be multiplied with the number of children to get a rough estimate the total time to find and resolve conflicts for a node.

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 we have

[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'

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., $(B, C, D) \rightarrow A$. Other nodes E, F, G lead a user to infer node A, i.e. $A \rightarrow (E, F, G)$. 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 9, we see that up to 10 nodes, the time to check policy consistency is under one second.

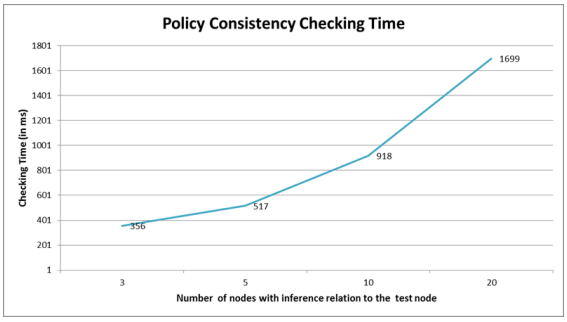

Figure 9. Policy consistency checking time.

V. Related Work

The problem of supporting access policies on hierarchical data elements is studied by some researchers. Most closely related to our research is the work by Jajodia et al. [6], which proposes some strategies for conflict resolution and authorization propagation. One difference of our work is that we consider inference relations in addition to hierarchical relations for conflict resolution. Also, Jajodia et al. only consider static resolution of conflicts and do not consider situations where access control rules themselves may change during execution.

Inference attacks are common against statistical databases [7], [12]. Results of statistical attacks can either contain probabilistic information or information that leads to inference with certainty. A type of inference attack was highlighted by De Jonge in his paper on data inference by results of queries seeking average of subsets of data. His work shows how datasets can be varied over the cardinality of the data to reveal valuable information using the average [7]. K-anonymity is one of the popular methods employed to protect databases against inference attacks. The method uses a configurable parameter k to determine the minimum number of entries which should match a query for any result to be returned [17]. This techniques makes sure that results to queries pointing to less than k items will not be returned, thus protecting individual information items. Another technique, called l-diversity, makes sure that the k or more data items returned as query results have at least l different values [11]. Li et al. proposed the concept of t-closeness in which they proposed that the l different values should be separated at least by a parameter t to ensure a uniform spread of the returned results [9]. All these popular methods have an assumption that a user either has access to all the sensitive data or they have no access to sensitive data.

Researchers have proposed various methods to reduce an attackers access to data that is highly correlated to protected data. Chang et al. used a Bayesian network model to determine these inference relations and then reduce the quality of non-sensitive information to reduce the impact of inference attacks [4]. This protection will have less impact in case of an attacker who has legitimate access to some sensitive information which is highly correlated to the non-accessible protected information. Yip et al. proposed five strategies for building these inference relations considering different ways in which sensitive and non-sensitive data sets interact [20]. Association of data types is another problem pointed out by Lunt [10]. The main idea is that two or more data objects can be non-sensitive by themselves but may become sensitive when they are associated in some manner. For example, ‘employee name’ and ‘salary for a designation’ may be non-sensitive by themselves by become sensitive upon associating them together.

Some researchers have also developed approaches for handling inference issues during database design [5], [13], [15], [10]. Hinke proposed the use of conceptual structures to detect inferences in databases [5]. He used the graph structure to represent the application and showed how data can be inferred by traversing alternate paths between two nodes. His design was simple and used transitivity to detect inferences. Others proposed to detect and restrict inference attacks during query processing [8]. Thuraisingham et al. developed methods for detecting inference relations and presenting them the system administrator in addition to the above two [18].

Several projects have developed support for detecting inferences related to duplicated data protected using inconsistent policies. Stoica et al. implemented a model where their security engine would detect ontologically equivalent

XML data elements which are replicated [16]. In that work, equivalence was established manually. Yang et al. proposed an inference engine to semantically match replicated XML data in large distributed databases and check inconsistency in their security classifications [19].

Previous approaches on checking access control rules have similarities to our approach but the main distinction is that we check inconsistency in access control rules between different data elements from which protected data elements can be inferred as opposed to the other methods, which try to detect replicated data. Another important distinction is that our approach is not limited to XML data and considers different types of data elements in the databases.

VI. 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.

Acknowledgment

This research is supported in part 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] eXtensible Access Control Markup Language (XACML). *www.oasis-open.org/committees/xacml/*.

[2] i2b2 web client. *https://www.i2b2.org/webclient/*.

[3] Sun XACML Policy Engine. *sunxacml.sourceforge.net*.

[4] L. Chang and I. Moskowitz. A bayesian network schema for lessening database inference. In *CIMCA01*, 2001.

[5] T. Hinke. Inference aggregation detection in database management systems. In *IEEE Symposium on Security and Privacy*, 1988.

[6] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26:214–260, June 2001.

[7] W. D. Jonge. Compromising statistical databases responding to queries about means. In *ACM Transactions on Database Systems*, 1983.

[8] T. Keefe, M. Thuraisingham, and W. Tsai. Secure query processing strategies. *IEEE Computer*, 22:63–70, 1989.

[9] N. Li and T. Li. t-closeness: Privacy beyond k-anonymity and l-diversity. In *IEEE 23rd International Conference on Data Engineering*, 2007.

[10] T. F. Lunt. Aggregation and inference: Facts and fallacies. In *IEEE Symposium on Security and Privacy*, 1989.

[11] A. Machanavajjhala, J. Gehrke, and D. Kifer. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1, March 2007.

[12] S. Mandujano. Inference attacks to statistical databases: Data suppression, concealing controls and other security trends. *Aleph Zero online magazine*, 23, April-May 2000.

[13] M. Morgenstern. Security and inference in multilevel database and knowledge base systems. In *ACM SIGMOD*, 1987.

[14] S. Murphy, G. Weber, M. Mendis, H. Chueh, S. Churchill, J. Glaser, and I. Kohane. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). *Journal of the American Medical Informatics Association*, 17(2):124–130, March-April 2010.

[15] G. Smith. Modelling security relevant data semantics. In *IEEE Symposium on Security and Privacy*, 1990.

[16] A. Stoica and C. Farkas. Ontology guided xml security engine. *Journal of Intelligent Information Systems*, 23:209–223, 2004.

[17] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10:557–570, 5 2002.

[18] B. Thuraisingham. The use of conceptual structures for handling the inference problem. *Database Security V: Status and Prospects*, pages 214–260, 1992.

[19] L. Yang, X. Junmo, and L. Jing. Protecting xml databases against ontology-based inference attack. In *International Conference on Computational Intelligence and Security*, 2007.

[20] R. Yip and K. Levitt. Data level inference detection in database systems. In *IEEE Computer Security Foundations Workshop*, 1998.