Constraint Graphs as Security Filters for Privacy Assurance in Medical Transactions

George Mathew, Zoran Obradovic
Center for Data Analytics and Biomedical Informatics
Temple University, Philadelphia, PA
{George.Mathew, Zoran.Obradovic}@temple.edu

ABSTRACT

We model transactions as exchange of graphs and propose constraint graphs as transaction filters for attribute-based transformations in clinical settings. An informal representation and working of constraint graphs is presented. A workbench was developed using a real world EMR system to demonstrate the concept. Results of our experiments using constraint graphs are consistent with published benchmark results.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; J.3 [**Life and Medical Systems**]: Medical Information Systems; K.4 [**Computers and Society**]: Privacy

Keywords

Medical informatics, Privacy

1. INTRODUCTION

Transactions involving patient data can be 1) administrative 2) financial 3) analytical or 4) clinical in nature. These transactions involve raw attributes that form a subset of the patient record. In another form of transaction in distributed data mining, raw data is not disclosed, but statistics about the attributes are passed along [1]. We introduce constraint graphs as a mechanism to enforce restrictions on data attributes in both these types of transactions.

Graph databases [2] are emerging as a viable alternative to relational models. The structure of most real life domain models is graph. Hierarchical structures, ad hoc structures, dynamic structures and semi-structured data are hard to be maintained by relational databases. Graph databases provide the agility to accommodate changes in underlying structures. In this study, we explore constraint graphs as a mechanism to control security in application-level transactions. Constraint graphs gateway is a service layer in a multi-tiered pipeline of services (see Figure 1).

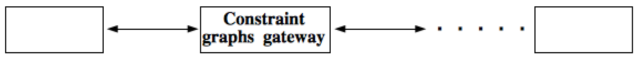

Figure 1: Constraint graphs service in a pipeline of services.

2. RELATED WORK

Differential Privacy [3] was designed for data analysis and not data transactions. In our case, for most transactions, raw data has to be shipped outside the system as opposed to differential privacy, where no data leaves the population database. Privacy Integrated Queries (PINQ) [4] is a data analysis platform that was designed to provide access to underlying data sets through a declarative language (LINQ) and provides formal guarantee of differential privacy. Our approach is similar. Constraint graphs enforce site-specific restrictions on data in transit.

3. PROTOTYPE GRAPHS AND CONSTRAINT GRAPHS

GraphQL [5] deals with graphs as the unit of operation. We model graphs as the currency for transactions. Pattern graphs in GraphQL are extended to prototype graphs and forms the basis for constraint graphs. Prototype graphs can be used for checking conformance on a given set of graphs. Constraint graphs are prototype graphs used specifically for enforcing transaction-based restrictions. The predicates on the attributes enforce the restrictions. Constraint graphs are useful in three ways. 1) To identify a matching subgraph in a given transaction graph 2) for masking specific attributes in selected graphs and 3) for data cleansing by applying targeted transformations on specified attributes.

In GraphQL, a graph motif consists of node and edge specifications. To extend graph motif, we define "general expression" ("genex") as generalization of the regex quantifier metacharacters. The genex tokens are shown in Table 1.

TABLE I. LIST OF GENERAL EXPRESSION TOKENS

Token	Interpretation
?	One item satisfies the condition.
+	At least one item satisfies the condition.
{n}	Exactly n items satisfy the condition.
{n,m}	At least n and ≤ m items satisfy the condition.
[ ]	One item in square brackets satisfies the condition.

The following is a graph motif:

```
graph G {   node v1;
            node v2;
            edge e(v1,v2);  }
```

When the graph motif has a tag followed by an optional genex, it is called a "genex motif". The following genex motifs A & B will match graph motif G. Genex motif C will not match motif G.

```
graph A { node x +;     graph B { node u +;     graph C {
          node y ?;}              node v +;          node u {3};}
                                  edge (u,v);}
```

A prototype graph is a genex motif with a set of predicates based on the tags. The following graph P is a prototype graph.

```
graph P {   node x + where x.sex = ‘m’;
            node y ? where y.temp > 99; }
```

A constraint graph is used for the purpose of filtering out (or modifying) specific attributes or relations in a set of input graphs for transactions. A constraint graph is a genex motif or a prototype graph and a set of actions from the following 3 forms:

```
exists <tag>[.<attribute>]
eliminate <tag>[.<attribute>]
substitute <tag>[.<attribute>] =~ /<transform>/
```

3.1 ‘exists’ action

A constraint graph with ‘exists’ action is a Boolean mechanism to qualify a transaction. If all the ‘exists’ conditions are satisfied, the transaction is unqualified. Otherwise, the transaction is qualified. For example, consider the constraint graph C:

```
graph C {  node u + exists u.name;
           node v + exists v.address;
           node w + exists w.dob;
           node x + exists x.id;  }
```

The attributes name, address, date of birth and id together in one transaction makes it unqualified.

3.2 ‘eliminate’ action

In the eliminate action, specified attributes are eliminated from matching graphs. Each eliminate action is independent of each other. For example,

```
graph E {   node x + eliminate x.dob;
            node y + where y.country = “USA” eliminate y.id; }
```

3.3 ‘substitute’ action

The format for substitute action is:

```
substitute <target>[.<attribute>] =~ /<transform>/
```

/<transform>/ can be a function or a pattern followed by a replacement pattern. If it is a function, it takes the format /fun()/. A pattern replacement takes the form /<original>/<replacement>/. Here, <original> is replaced by <replacement>. For example:

```
graph S {  node u + substitute  u.id =~ /hash()/;
           node v + substitute v.dob =~ /dob_transform()/;
           node x + substitute x.name =~ /Mike/Michael/;   }
```

3.4 Combining actions

Sometimes it is necessary to combine actions to perform more interesting transformations. For example, when data records in transit have ‘name’ attribute, it is possible to de-identify ‘name’ and add another attribute to replace ‘name’. Suppose this action is referred to as ‘add’. The following constraint graph represents a possible syntax to capture this idea:

```
graph N {  node u + exists u.name;
           node u + eliminate u.name;
           node u + add u.identifier =~ de-identify(u.name);  }
```

4. EXPERIMENTS

The first two sets of experiments were done in interactive mode, while the third set was performed in batch (non-interactive) mode.

4.1 Workbench in OpenEMR

The OpenEMR [6] electronic health information system was used as the platform for experiment. We implemented the constraint graphs server using SOAP Web Services protocol (see Figure 2).

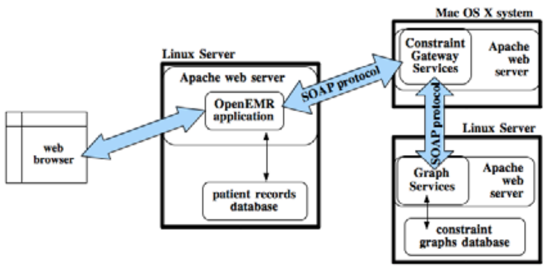

Figure 2: Details of Constraint graph gateway implementation.

We introduced a user interface (Workbench) for a set of transactions that could be initiated in the context of a patient. The Workbench allows for the selection of attributes of interest and associate a transaction type. We defined three transaction types: 1) financial 2) administrative and 3) research. Constraint graphs for these transaction types were defined and stored in a neo4j graph database. Since neo4j supported JAVA natively, we used a SOAP-based service for exchange between JAVA and PHP. Transformation rules were implemented as PHP functions.

When a transaction is initiated from the Workbench, a graph based on the selection is generated and sent to the gateway service. Based on the transaction type, the set of constraint graphs for that type is retrieved and applied on the data graph. An example of a constraint graph for financial transactions is:

```
graph C {  node x + substitute x.ss =~ /ss_transform()/; }
```

The resultant graph is also passed back to the workbench. In a working transaction implementation, the transformed data graph will be forwarded to the external agency associated with the transaction. We verified the working of various constraint graphs with actions using different data graphs within the Workbench.

4.2 Combining Actions

To illustrate the working of combining actions (sec 3.4), we used Bloom filters in a ‘research’ transaction within the Workbench. We specified a constraint that ‘name’ and ‘address’ attributes be eliminated and a new attribute ‘identifier’ be added. A function de_identify() with ‘name’ and ‘address’ as inputs generated the ‘identifier’. The function concatenated ‘name’ & ‘address’ and the result string was run through Bloom Filters to generate 24 hash bits in a 100-bit array. The hash bit positions were sorted in ascending order and combined to generate an identifier. We defined a constraint graph for ‘research’ transaction using the function de_identify(). Research transactions were initiated to trigger the constraint graph filtering. The results were verified to be correct.

4.3 Local Attribute Policy Enforcement

These set of experiments, based on the work published on Distributed ID3-based Decision Tree (DIDT) [1], were used to enforce local data attribute policies using constraint graphs. The DIDT algorithm builds a decision tree using statistics about data from distributed sites. The algorithm iteratively builds the nodes. There are two stages in each iteration. In stage one, a global schema for data is generated using metadata from the sites. In stage two, the node to be split is decided using crosstable matrices of attributes in the global view. The DIDT algorithm starts with a query to zoom in on attributes of interest. The query is mediated by a Clearing House (CH) and forwarded to sites of interest in the network. Individual sites identify instances satisfying the query and forward metadata about the instances to the CH. The CH aggregates the metadata to create a global view, which is used by sites to generate crosstable matrices for attributes. The crosstable matrix for an attribute is a representation for the number of instances that has a specific value and belonging to a specific class. These matrices are used by CH in calculating gain to select the node to be split in the decision tree. In each stage, there may be local data policies that prohibit the disclosure of attributes. The experiments and results published in the cited work under the section "*Learning from Multiple Sites when some sites constrain certain attributes*" are use cases of this scenario. We used our recent results [1] as the benchmark.

The baseline experiments used SPECT heart data set of patients from the UCI machine-learning repository [7]. The data set consists of summary features of 267 cardiac Single Proton Emission Captured Tomography (SPECT) images. The patients are classified as normal or abnormal. The 22 binary attributes were labeled a1, a2, . . . , a22 and classes were labeled c0 & c1. Duplicates with all 0's were eliminated. The resulting 256 data instances were distributed randomly to 16 sites, each site getting 16 instances. The baseline query was $a3 \wedge a5 \wedge a8$ to select data instances with positive values for a3, a5 and a8. During the process, attributes were constrained by suppression from being disclosed. When we re-enacted the experiments, attribute constraints were implemented using constraint graphs with 'eliminate' actions. The layout of gateway services and interaction with CH is shown in Figure 3.

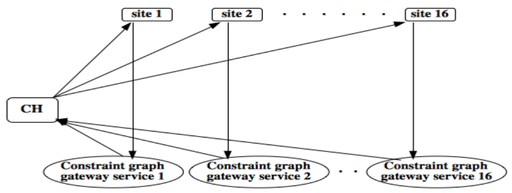
site 1
site 2
site 16
CH
Constraint graph gateway service 1
Constraint graph gateway service 2
Constraint graph gateway service 16

Figure 3: Constraint Graph Gateway Used for Local Attribute Policy Enforcement

Attributes a13 and a10 were independently blocked from 4,12 & 16 sites in 3 consecutive experiments and DIDT was constructed using 16 leave-one-site-out cross-validations. Results are shown in Table 2.

TABLE II. RESULTS OF CONSTRAINING AN ATTRIBUTE

attribute blocked	# of sites blocking	cross-validation	correctly classified	accuracy
a13	4,12,16	16	36/37	97.30%
a10	4,12,16	16	32/37	86.47%

These results are consistent with the published benchmark results, thus verifying the working of the actions. The minor differences in correctly classified instances in Table 2 vs. published benchmark results are due to the randomness of data distribution to the sites and consequent aberration in cross-validation results.

Next set of experiments was using exclusive and inclusive constraints enforcements. In both cases, a pair of attributes was excluded from a fixed number of sites. In exclusive mode, each attribute was excluded from specified number of sites with no overlap; while in inclusive mode, the pair of attributes was simultaneously excluded from a given number of sites. Results are shown in Table 3.

TABLE III. RESULTS OF COMBINED BLOCKING OF a13 & a10

type of blocking	# of sites blocking	cross-validation	correctly classified	accuracy
Exclusive	8	16	33/37	89.12%
Inclusive	4	16	33/37	89.12%

These results are also consistent with the corresponding published benchmark results, thus verifying the working of constraint graphs

5. CONCLUSIONS

We presented graph databases as viable store for clinical data and constraint graphs as a filtering mechanism to enforce local data policies; thus, assuring privacy in medical data transactions. We implemented a workbench in an electronic health records system to demonstrate the concept. In the case of constraining statistics about data, we demonstrated the working of constraint graphs as a successful filtering mechanism. Future work directions include incorporating more actions, sophisticated general expressions in actions and formal specification of actions.

6. ACKNOWLEDGEMENT

This project is funded in part under a grant with the Pennsylvania Department of Health. The Department specifically disclaims responsibility for any analyses, interpretations, or conclusions.

7. REFERENCES

[1] Mathew, G and Obradovic, Z. 2011. A Privacy-Preserving Framework for Distributed Clinical Decision Support. In *Proceedings of the 1st IEEE International Conference on Computational Advances in Bio and medical Sciences.* Feb 2011, Orlando, FL. DOI=http://dx.doi.org/10.1109/ICCABS.2011.5729866

[2] Aggarwal, C. C., and Wang, H. 2010. *Managing and Mining Graph Data*. Springer, NY, USA.

[3] Dwork, C. 2006. Differential Privacy. In *Proceedings of 33rd International Colloquium on Automata, Languages and Programming*, July, 2006. Venice, Italy, pp. 1-12.

[4] McSherry, F. D. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 35th SIGMOD International Conference on Management of Data*, Providence, RI. pp. 19-30. DOI=http://dx.doi.org/10.1145/1559845.1559850.

[5] He, H., and Singh, A. K. 2008. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. Vancouver, Canada. DOI=http://dx.doi.org/10.1145/1376616.1376660.

[6] Home page for Open EMR software, 2010. http://www.oemr.org.

[7] SPECT Heart Data Set: Available at – http://archive.ics.uci.edu/ml/datasets/SPECT+Heart