REDACT: A Framework for Sanitizing RDF Data

Jyothsna Rachapalli, Vaibhav Khadilkar, Murat Kantarcioglu, Bhavani Thuraisingham
The University of Texas at Dallas
Richardson, Texas, USA
{jxr061100, vvk072000, muratk, bxt043000}@utdallas.edu

ABSTRACT

Resource Description Framework (RDF) is the foundational data model of the Semantic Web, and is essentially designed for integration of heterogeneous data from varying sources. However, lack of security features for managing sensitive RDF data while sharing may result in privacy breaches, which in turn, result in loss of user trust. Therefore, it is imperative to provide an infrastructure to secure RDF data. We present a set of graph sanitization operations that are built as an extension to SPARQL. These operations allow one to sanitize sensitive parts of an RDF graph and further enable one to build more sophisticated security and privacy features, thus allowing RDF data to be shared securely.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features; D.4.6 [**Operating Systems**]: Security and Protection—*Access Control*

Keywords

SPARQL; Sanitization; Access Control; RDF; Provenance

1. INTRODUCTION

RDF is the key to the Semantic Web vision of a global database formed by seamless integration of various data sources. Although such data integration will lead to an unprecedented wealth of knowledge, RDF does not provide any means to securely share data. We illustrate this with a *motivating scenario: Assured Information Sharing (AIS)* refers to organizations sharing information while enforcing policies/procedures so that the integrated data can be queried securely to extract nuggets. An AIS system integrating data sources from agencies such as the Army, Navy, Air Force, *etc.* is critical as it may contain sensitive information. While the different agencies have to share data, they need to do so in a secure manner by sanitizing the sensitive data. Although RDF provides an elegant solution for data integration, it falls short in providing a means for sanitization. We therefore present a tool REDACT (**R**df **ED**iting **A**nd **C**oncealing **T**ool), which will help protect data privacy and enable secure sharing of sensitive RDF data by means of RDF graph sanitization operations. **Our Contributions**: (a) We extend SPARQL with a set of fundamental graph sanitization operations to secure RDF. (b) We formalize this language extension using denotational semantics, which can be found in our technical report [4]. (c) We present empirical results showing the performance of the sanitization operations.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

2. SPARQL LANGUAGE EXTENSION FOR GRAPH SANITIZATION

A sensitive resource in an RDF graph can be a data value or a resource node, a class of data values or resource nodes, an edge connecting two nodes, a path containing multiple nodes connected by edges, *etc.* We present a set of corresponding graph sanitization operations namely, Sanitize Node (SNode), Sanitize Edge (SEdge), Sanitize Path (SPath), *etc.* The SPARQL sanitization queries containing the sanitization operations when run on an RDF graph, transform only the sensitive resource described by the LHS of the Sanitization operation (as shown in example queries Q1-Q6) and the remaining triples of the graph stay unchanged. In the following, we illustrate our sanitization operations on an example provenance [3] RDF graph from healthcare domain captured using Open Provenance Model Vocabulary [2]. The purpose of the SNode operation is to sanitize and protect sensitive data values or resources such as SSN, Secret service agents, *etc.* An example SPARQL query Q1 that sanitizes the object part of a sensitive triple and transforms the SSN value from "123-45-678" to "XXX" is shown in Figure 1. If one needs to conceal all SSN values of a graph, one may use the variation of SNode shown in Q2, where the sensitive resource(s) is represented by a suitable triple pattern. However, if one needs to conceal the SSN values of a class of individuals, such as Physicians, then one may use the variation of SNode shown in Q3.

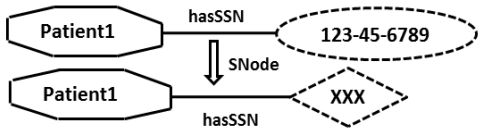

Figure 1: SNode

```
Q1: SELECT ?x ?y ?z
    WHERE {(Patient1 hasSSN "123-45-678") SNode (?x ?y ?z)}

Q2: SELECT ?x ?y ?z
    WHERE {(?s hasSSN ?o) SNode (?x ?y ?z)}

Q3: SELECT ?x ?y ?z
    WHERE {(?s rdf:type Physician ?s hasSSN ?o) SNode (?x ?y ?z)}
```

The operation SEdge is designed to protect an edge along with its two nodes. The query Q4 sanitizes a sensitive relationship by hiding the edge along with its two nodes, and is illustrated in Figure 2. In addition, SEdge supports sanitization of a sensitive relationship of a class of individuals as shown in query Q5. The operation SPath is designed to protect a path (subgraph) containing multiple nodes connected by edges, and can be represented using a path pattern [1]. Query Q6 uses SPath to hide the provenance of PatientFile1. This query consistently sanitizes the sensitive subgraph represented by the path pattern on the LHS of the SPath operation, and the rest of the graph remains unchanged.

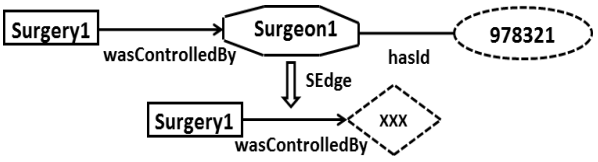

Figure 2: SEdge

```
Q4: SELECT ?x ?y ?z
    WHERE {(Surgeon1 hasId 978321) SEdge (?x ?y ?z)}

Q5: SELECT ?x ?y ?z
    WHERE {(?s rdf:type Surgeon. ?s hasId ?o) SEdge (?x ?y ?z)}

Q6: SELECT ?x ?y ?z
    WHERE {(PatientFile1 [wasDerivedFrom]+ ?o) SPath (?x ?y ?z)}
```

3. ARCHITECTURE AND PERFORMANCE

In Figure 3, we show the architecture of a prototype system that uses graph sanitization to protect underlying sensitive RDF data. A querying user writes a query (SPARQL) and submits it to the user interface. The user interface forwards it to the SPARQL query engine and also forwards the query along with the user information to the policy evaluator (as depicted by 2b). Once the query engine receives the query, it performs computation on the data stored in the RDF store. It then returns the resulting graph G_q to the policy evaluator (as depicted by 5). The Policy evaluator now computes a set of disjoint applicable policies from the original policy set based on the three aforementioned inputs it receives. Policies are called disjoint when the resources protected by them are non-overlapping. We use disjoint policies in order to avoid side effects such as concurrent sanitization. Next, an optimal set of policies is obtained from the disjoint applicable policy set, under the policy author defined risk and utility constraints, using a dynamic programming algorithm [4]. An optimal set of policies guarantees that one can obtain the right mix of usability after sanitization, while maintaining the risk value below the specified threshold. Subsequently, the policies in the optimal set are parsed and transformed into corresponding SPARQL sanitization queries such as SNode, SEdge, *etc.* Once the list of sanitization queries is obtained, we then apply them on to the RDF graph and return the sanitized graph to the querying user. We now present results of experiments that were conducted to evaluate the performance and scalability of the sanitization operations. The datasets used for evaluation were Twitter and U.S. Securities and Exchange Commission (SEC), each containing at most 3M and 1.8M triples respectively. It can be seen from Figure 4 that the time to

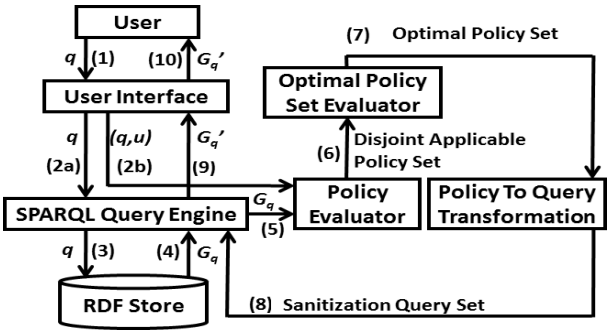

Figure 3: Prototype System Architecture

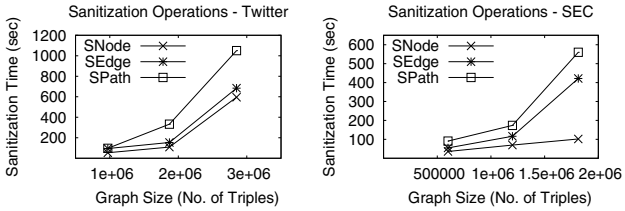

Figure 4: Performance on different datasets

execute the sanitization operations increases as the dataset size increases. However, such an increase is not linear as it involves a multitude of variable conditions such as the connectivity of the input resource, nature of the resource being protected by the policy, *etc.* Additionally, operation SNode, requires a shorter execution time when compared with more complex operations such as SPath.

4. CONCLUSION

Using REDACT, Semantic Web applications can secure underlying RDF data with graph sanitization operations. Further, they can build more sophisticated security features, as illustrated with the prototype system architecture. Finally, the formal underpinning using denotational semantics makes REDACT a natural extension of SPARQL.

5. ACKNOWLEDGMENTS

This work was partially supported by Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265 and FA9550-12-1-0082, National Institutes of Health Grants 1R0-1LM009989 and 1R01HG006844, National Science Foundation (NSF) Grants Career-CNS-0845803, CNS-0964350, CNS-1016343, CNS-1111529, CNS-1228198 and Army Research Office Grant W911NF-12-1-0558.

6. REFERENCES

[1] S. H. Garlik, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language.

[2] O. Hartig and J. Zhao. Provenance Vocabulary Core Ontology Specification. *Change*, (July), 2010.

[3] L. Moreau. The foundations for provenance on the web. *Found. Trends Web Sci.*, 2, Feb. 2010.

[4] J. Rachapalli, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. REDACT: A Framework for Sanitizing RDF Data. `http://goo.gl/wT38D`.