SPARQL Access Policies

Abstract

The lack of a (portable) access control language has been repeatedly identified as a major obstacle to Semantic Web adoption by industry. Development of the Semantic Web stack as a set of open standards has encouraged academic and public service involvement, indirectly promoting a set of use cases which did not require access control. The need to demonstrate the effectiveness of data access has futher discouraged developers from working with data which could not be publicly shared. Where access control was required, it was usually implemented at a resource level; those with the proper credentials could see all of a document; those without could see none of it. The use of graph patterns allows us to select portions of graphs, potentially spanning multiple resources. Tying access control policies to the graph patterns allows us to map pricipals and roles to fine-grained portions of graphs. Industry is just now paying attention to the Semantic Web, which means it's really time to deliver robust and predictable security.

State of the Art

Within the Semantic Web stack, logic languages such as CWM were used to implement simple rule-based policies. The AIR language extended this to provide fallback policies and user feedback for some debugging help (http://tw.rpi.edu/proj/tami /AIR_Language_Formalization). Much of the data on the Web is available through relational databases (RDBs). RDBs have decades of development in access control expressions, the most interesting of which are vendor-specific extensions for e.g. row and column-based access control.

In the XML world, XACML has enjoyed relatively wide deployment [1]. XACML involves a complex protocol of Protocol Decision Points and Protocol Enforcement Points, enabling a decoupling of policy from data. XACML profiles map closed content models to problem domains, much as one ontologies to problem domains in the Semantic Web. Likewise, in clinical settings, I2B2 [2] has promised a sufficient balance between secrecy and utility that Institutional Review Boards (IRBs, who ultimately decide who can access data) routinely grant access to data which researchers actually find useful.

Problem Statement

There are countless reasons to keep secrets, but some of the most intuitve and accessible scenarios come from health care. Inappropriate access to health care data reeks of injustice, and rallies the privacy-minded to great acts of legislation. Clinical records are typically mandated by federal laws (e.g. HIPAA), state, municiple, institutional, and perhaps laboratory regulations. The decoupling of the policy from the data makes it very hard to confidentally assert compliance with all these levels of policy.

As a specific example, consider a simplicied clinical record with patient identity and prescriptions, and care giver information:

Database Data

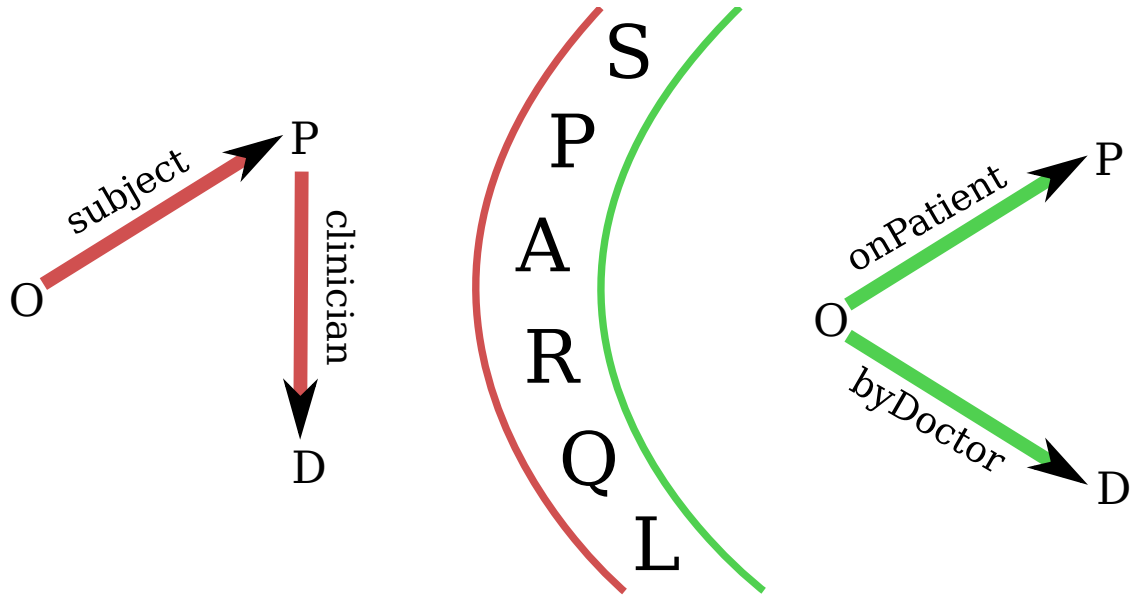
```
PREFIX :mydb <http://cityhospital.example/dbs>…
obs:2396 mydb:patient patient:218 .
obs:2396 mydb:doctor  doctor:85 .
doctor:85 mydb:name    "Dr. Jones" .
patient:218  mydb:patientName "Bob Smith" .
```

We may not want to make our e.g. research partners or patient medical record access use our home grown ontology which is really a slave to structural storage or acquisition paths. We can use SPARQL to CONSTRUCT a more pallatable form of the data:

Use of Popular Ontologies

```
   PREFIX :mydb <http://cityhospital.example/dbs>
CONSTRUCT { ?o a          :PatientObservation .
            ?o :patient   ?p .
            ?p foaf:name  ?pName .
            ?p :takes     ?takes .
            ?o :doctor    ?d .
            ?d foaf:name  ?dName }

    WHERE { ?o mydb:patient ?p .
            ?o mydb:doctor  ?d .
            ?d mydb:name    ?dName .
            ?p mydb:patientName ?pName .
          }
```



This is a pretty straight-forward use of SPIN to map in-house data to a format we care to share with the world, our research partners or the folks on the other side of the lab.

This data includes identification and medication information, the exposure of both of which are regulated by potentially different entities.

Policy "Annotation"

The SPIN rule above injects several pieces of delicate information into a shared representation of clinical data. That data is specifically bound by mathing the WHERE clause

against the input data. How can we add some caveats to that binding to moderate the disseminated information? First we will need some database of credentials to which we can tie access permissions.

Trivial User Database

Our dateabase can contain IP address, client-side SSL certificates, kerberose keys, etc. Suppose, for a very simple example, we associate identites with usernames and passwords:

```
GRAPH <ACCESS> {
    <user1> acls:username "Sue" ;
            acls:password "suezpass" .
    <aclA> acls:entitles <user1> , <user2> ;
           acls:toSee acls:identity , acls:labResults .
}
```

Protocol-enhanced Query

Now we have to extend SPARQL a bit. If the protocol pastes information into the query, say, substituting `$USER` and `$PASS` in

```
GRAPH <ACCESS> {
    ?_requestor acls:username $USER ;
                acls:password $PASS .
}
```

, we bind `?_requester` to the user who made the request. We can now predicate the binding of sensitive information on particular entitlements of the requester.

```
OPTIONAL {
    ?p   ppl:familyName ?pName .
    GRAPH <ACCESS> {
        ?acl acls:entitles  ?_requester ;
             acls:toSee  acls:identity .
    }
}
OPTIONAL {
    ?p   obs:medication ?takes .
    GRAPH <ACCESS> {
        ?acl acls:entitles  ?_requester .
            acls:toSee  acls:medication
    }
}
```

This says that there must be some ACL which permits the user to see identity information, or ?pName will not be not bound. Likewise, `?_requester` must be entitled to see medication information, or ?takes is not bound. (The terms `identity` and `medication` were taken from the XACML Medical Data Profile.)

Per the CONSTRUCT rules of SPARQL, any triples with these variables will not be emitted if these variables are not bound. This has exactly the behavior we want: if the user is allowed to see identify information, they'll get triples like `patient:218 foaf:name "Bob Smith"` Independently, if they are allowed to see medication, the input graph will contain medication assertions like `patient:218 :takes "Illudium Phosdex"` . Noting the ACL above that permits `<user1>` to see identity and labResults information but not medication, any query she will not be able to ask

questions about medication will be able to ask questions like:

```
   PREFIX : <http://cityhospital.example/dbs/observations>
   PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
 WHERE { [ a :PatientObservation ;
           :doctor  [ foaf:name  "Dr. Funkenstein" ] ;
           :patient [ foaf:name  ?pName ]
         ] . }
```

Tailored View Example

Assembling the pieces above, we see a rule which creates a graph the contents of which depend on the viewer:

```
   PREFIX :obs <http://cityhospital.example/dbs/observations>
   PREFIX :ppl <http://cityhospital.example/dbs/people>
   PREFIX :med <http://cityhospital.example/dbs/medication>
   PREFIX :acl <http://cityhospital.example/dbs/acls>
CONSTRUCT { ?o a          :PatientObservation .
            ?o :patient   ?p .
            ?p foaf:name  ?pName .
            ?p :takes     ?takes .
            ?o :doctor    ?d .
            ?d foaf:name  ?dName }

    WHERE {
        GRAPH <ACCESS> {
            ?_requestor
                acls:username $USER ;
                acls:ip       $IP ;
        }
        ?o obs:patient ?p .
        ?o obs:doctor  ?d .
        ?d ppl:name    ?dName .
        OPTIONAL {
            ?p   ppl:familyName ?pName .
            GRAPH <ACCESS> {
                ?acl acls:entitles  ?_requester ;
                     acls:toSee  acls:identity .
            }
        }
        OPTIONAL {
            ?p   obs:medication ?takes .
            GRAPH <ACCESS> {
                ?acl acls:entitles  ?_requester .
                    acls:toSee  acls:medication
            }
        }
    }
```

One nice feature of this strategy is that the sensitive data is specifically in the context of the policy use to control access to it. This makes auditing and updating much easier than with decoupled systems. Probably the most interesting feature is that both the ACLs and the exposed information can be arbitrarily complex graphs. For instance, a person acting in the role of researcher or care-giver from either of two instituions (known by IP address) may see ADHD medications, unless the prescription was in New York State and the medication is also classed as an antidepressant.

```
    WHERE {
      … OPTIONAL {
```

```
        ?prescription pres:medication ?takes ;
                      pres:start ?start ;
                      pres:end ?end .
        GRAPH <ACCESS> {
            ?userrole roles:user $USER ;
                      roles:role ?role
            FILTER (?role = "researcher" || ?role = "care-giver")
            FILTER (?IPADDR = "10.2.3.4" || ?IPADDR = "192.172.5.6")
        }
        ?takes meds:application "ADHD" .

        # OPTIONAL { …?inst… } !BOUND(?inst) is an idiom for negation as failure in SPARQL 1.
        OPTIONAL { ?prescription obs:institution ?inst .
                   ?inst addr:city "NY" .
                   ?takes meds:application "antidepressant" }
        FILTER (!bound(?inst))
    } …
}
```

This example is much more complex than the introductory query, but requires no protocol extension or other infrastructure changes, only a richer access database and exposure rules.

Execution Strategies

The policy expression described above may be reallized multple ways. Given that it is a SPARQL CONSTRUCT, it could be used to create a materialized view. The cost of that, in addition to the usual concurrency issues, is that a view must be created for each possible accessor. More practical is to use the CONSTRUCT as a simple declaration of the policy, or as a template for query rewriting.

Issues

- *Cardinality* — the solutions are multiplied by the numer of ways the querier is granted access. This can be addressed with subselects, but that's pretty awkward.
- *Demarkation* — using variables like $CAPITALS as above means the rules can be validated verbatim. A syntactically invalid form like ~USER would ensure that the query wasn't exected without proper substitution.

References

- [1] S. Godik et al., “eXtensible Access Control Markup Language (XACML) Version 1 . 0,” 2003.
- [2] S. N. Murphy et al., “Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2).,” Journal of the American Medical Informatics Association, vol. 17, no. 2, pp. 124-130, 2010.