Using RDF Metadata To Enable Access Control on the Social Semantic Web

James Hollenbach, Joe Presbrey, and Tim Berners-Lee

Decentralized Information Group, MIT CSAIL,
32 Vassar Street, Cambridge, MA, USA, 02139
{jambo,presbrey}@mit.edu
{timbl}@w3.org
http://dig.csail.mit.edu

Abstract. The structure of the Semantic Web gives users the power to share and collaboratively generate decentralized linked data. In many cases, though, collaboration requires some form of authentication and authorization to ensure the security and integrity of the data being generated. Traditional authorization systems that rely on centralized databases are insufficient in this scenario, since they rely on the existence of a central authority that is not available on the Semantic Web. In this paper, we present a scalable system that allows for decentralized user authentication and authorization. The system described supports per-document access control via an RDF metadata file containing an access control list (ACL). A simple interface allows authorized users to view and edit the RDF ACL directly in a Web browser. The system allows users to efficiently manage read and write access to linked data without a centralized authority, enabling a collaborative authoring environment suited to the Semantic Web.

Key words: Access Control, Semantic Web, FOAF+SSL, RDF, Web Access Control, Social Web, Metadata Management, Collaborative Authoring

1 Introduction

In the Semantic Web community there is a growing interest in collaborative authoring of linked data. In most collaborative environments, though, there must be an adequate authorization system in place to restrict which users can view and edit certain data. When designing such a system for use on the Semantic Web, the following important factors must be taken into account:
Decentralized accounts. On the Semantic Web, data is stored on many machines in a decentralized manner. Requiring users to have separate accounts for each server that they edit data on would quickly become cumbersome; instead, having a single user identity would be ideal. Additionally, such a sign-on system should not be located on a centralized server. Instead, users should be able to maintain their own accounts on remote servers.

Potential for rule-based authorization. One of the greatest potential benefits of the Semantic Web is the power to reason over data provided by multiple sources. A good system should allow administrators to create rules for authorization, such as "Only my friends and friends of my friends can access my data".
Authentication speed. Even though a decentralized authentication mechanism will by design require that information be pulled down from a remote server, the speed of the authentication itself should still be at least comparable to traditional authentication mechanisms. To achieve this, the system should ideally only need to make one request to a remote server in order to complete authentication. Following this initial authentication, session caching should be used to prevent repeated requests to remote servers, making repeated authentication steps effectively identical in speed to traditional authentication methods.
Ease of maintenance. The system should allow authorized users to intuitively add and remove access for other users. Since the access control system is intended for data shared on the Semantic Web, any ACL should also be editable from a Web browser. This could potentially be accomplished using protocols such as WebDAV [1] or SPARQL/Update [2].

The system we have implemented allows for decentralized accounts, fast authentication, and easy maintenance of ACL metadata. Support for rule-based authorization is not currently included in favor of a simpler ACL-based implementation, but we will show that support for such reasoning could be added. The resulting system allows users to collaboratively edit linked data stored on the Web server.

The remainder of the paper discusses our experiences in implementing and assessing our system. In Section 2, we briefly introduce the prior work that our system is based on. In Section 3, we describe the back-end system that allows for decentralized authentication and ACL-based authorization. Section 4 describes the interface for viewing and editing access control files. Finally, Sections 5, 6, and 7 discuss the performance of our implementation and potential future work that would expand and improve on the present system.

2 Background

In this section, we briefly discuss the prior work related to our system and how that work affected our design.

2.1 RDF-based Access Control

This is not the first attempt at creating an RDF-based ACL system. Since 2002, the World Wide Web Consortium (W3C) has used an RDF-based ACL system [3] to control access to files on its servers. The W3C ACL system has proven very useful for managing access to W3C files, but it has a few limitations that we aim to improve on. First, though the actual rules granting access to a file are written in RDF, authentication and authorization are still handled by a central database. Therefore, multiple entities trying to use the W3C ACL system would either need to share or mirror the same database for authentication. Additionally, user information for the W3C ACL system does not necessarily identify the user with a URI. As a result, user information in the W3C ACL database cannot easily be shared with external entities without some prior knowledge of W3C users and groups.

Despite this, the W3C ACL system provides a strong base for developing a similar system for use on the wider Semantic Web. The major difference between the W3C ACL system and our system is the prominence of linked data and standardized protocols. By moving all user and group data onto the Semantic Web and using a common login mechanism, we allow for access rules to be shared more readily between different organizations.

2.2 Collaborative Authoring

We identified two potential options for allowing collaborative authoring of files in our system: WebDAV and SPARQL/Update. Both have distinct advantages and disadvantages. As far as support for linked data, SPARQL/Update is the clear choice: it supports insertion and deletion of small amounts of data from the target graph. WebDAV, on the other hand, is not explicitly intended for authoring linked data documents. Instead, when performing a WebDAV PUT, the entire document must be rewritten to the server. If there were no other limiting factors, SPARQL/Update would be the obvious choice.

However, due to our desire to produce a usable Apache [4] module, we had to rely on what currently exists. As of the writing of this paper, there is no Apache module that supports SPARQL/Update. However, WebDAV is available in the Apache module `mod_dav`, guaranteeing widespread support. For this reason, our current implementation uses WebDAV rather than SPARQL/Update to handle updates to the triple store and ACL metadata. In Section 6, we discuss how the system could easily be modified to instead use SPARQL/Update when such a module becomes available.

2.3 Decentralized Authentication

There exist two potential methods for performing decentralized authentication: OpenID [5] and FOAF+SSL [6]. While OpenID is a more established protocol, FOAF+SSL has a major advantage in that it links directly to a user's FOAF [7] file. The information in the user's FOAF profile (or whatever other information is stored at the URI) could potentially be used in rule-based authentication. OpenID, on the other hand, simply provides a decentralized ID that does not necessarily have any linked data associated with it.

Briefly, the FOAF+SSL protocol works as follows: a user's public encryption key is stored in both a certificate and a remote RDF file stored on a Web server. Additionally, the certificate is created with the URI for the location of the remote RDF file stored in the certificate's `Subject Alternative Name` field. When a user sends a FOAF+SSL certificate to a server, the server identifies the remote URI, pulls down the public key from that URI, and compares it with the public key in the user's certificate. If the keys match, the user is authenticated as the entity associated with the `(public key,remote URI)` pair described in the certificate. The server can then decide whether or not to authorize a user based on a set of rules–in our case, a simple access control list. By using self-signed certificates, FOAF+SSL provides a trust-based user network that does not rely on any central authority, which by design works perfectly in a Semantic Web environment.

3 Server Implementation

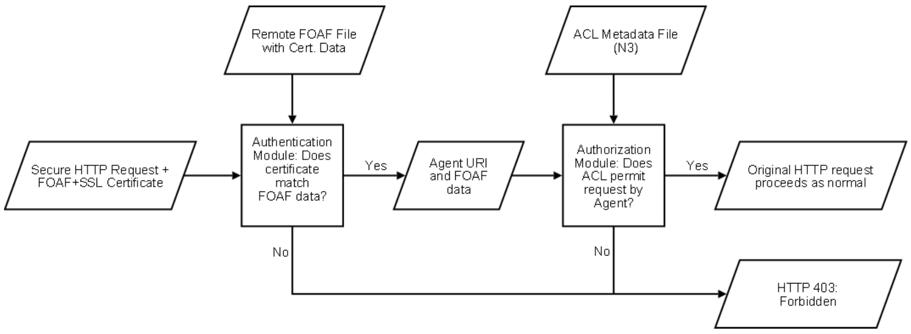

Remote FOAF File with Cert. Data
ACL Metadata File (N3)
Secure HTTP Request + FOAF+SSL Certificate
Authentication Module: Does certificate match FOAF data?
Yes
Agent URI and FOAF data
Authorization Module: Does ACL permit request by Agent?
Yes
Original HTTP request proceeds as normal
No
No
HTTP 403: Forbidden

Fig. 1. Flowchart depicting the authentication and authorization process. The authentication and authorization modules are implemented as Apache modules.

In this section, we describe the server-side implementation of our access control system. The server is implemented as two Apache modules: an authentication module (`mod_authn_webid`) and an authorization module (`mod_authz_webid`). The general flow of the authentication and authorization process is depicted in Figure 1.

3.1 Authentication

The authentication module implements the FOAF+SSL protocol to determine if the public keys in the user's certificate and FOAF file match. Authentication via this module is enabled in the server's Apache configuration with the `AuthType WebID` Apache directive. To perform authentication, the module checks for a URI in the `Subject Alternative Name` field of the certificate. Next, the certificate's signature is calculated using Apache's `mod_ssl` [8] module. If this process completes successfully, the user's remote URI is dereferenced to see if it contains matching certificate data using the Redland RDF library [9]. Finally, the SPARQL [10] query in Figure 2 is run. The query searches for all instances of a public key `?key` and its associated agent `?agent`. If the query returns a result

```
SELECT ?mod_hex WHERE {
    ?key rdf:type rsa:RSAPublicKey.
    ?key rsa:public_exponent ?exp.
    ?key rsa:modulus ?mod.
    ?key cert:identity ?agent.
    ?exp cert:decimal CERT_DECIMAL.
    ?mod cert:hex ?mod_hex.
}
```

Fig. 2. This SPARQL query finds all of the certificate data in the agent's remote FOAF file. Each set of certificate data consists of an agent (user ID) and an RSA key with an exponent and modulus.

that matches the key and agent provided in the user's certificate, the module authenticates the user as acting on behalf of the agent and returns control back to Apache.

3.2 Authorization

The authorization module makes use of an N3 [11] metadata file that contains an access control list (ACL). Metadata is currently stored on a per-directory basis, though the system could easily be modified to store ACL metadata on a per-document or even per-resource basis (see Section 6). The server directs clients to the ACL for a given file using the `link rel=meta` HTTP header [12]. Figure 3 shows a simple ACL used by the system. The ACL ontology used is the Web Access Control Ontology [13]. For a given rule, `acl:accessTo` defines a resource that access is being granted to, `acl:agent` and `acl:agentClass` define an agent or agent class (such as "any foaf:Person") as being granted access, `acl:mode` defines the set of modes that are granted to the agent or agent class, and `acl:defaultForNew` optionally defines the default access rules for new documents in a directory.

The three types of access that can be granted are `Read`, `Write`, and `Control`. Each type of access is bound to a specific set of HTTP methods as described in Table 1. The `Read` and `Write` modes control access to normal files stored on the server. `Control` is essentially a special form of `Write` access: having `Control` access to a file allows a user to edit the ACL metadata for that file. As with the authentication module, authorization is determined by running a set of SPARQL queries to determine if the user is granted access either as an agent or as a member of an agent class. If a user attempts to use an HTTP method that they are not permitted to use, they receive a 403 response from the server. For example, using the ACL file from Figure 3, if a user authenticated with the URI `http://www.example.com/foaf#me` tried to delete `foaf.rdf` with an HTTP DELETE request, the server would see that only `presbrey` has `Write` access and respond with `403 Forbidden`.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#> .
[]
  a acl:Authorization ;
  acl:defaultForNew <.> ;
  acl:accessTo <foaf.rdf> ;
  acl:agent <http://presbrey.mit.edu/foaf#presbrey> ;
  acl:mode acl:Control, acl:Read, acl:Write .
[]
  a acl:Authorization ;
  acl:accessTo <foaf.rdf> ;
  acl:agentClass <http://xmlns.com/foaf/0.1/Agent> ;
  acl:mode acl:Read.
```

Fig. 3. A simple ACL file. This file grants http://presbrey.mit.edu/foaf#presbrey Read, Write, and Control access to the file foaf.rdf, and Read access to foaf.rdf to all authenticated FOAF+SSL users. It also grants presbrey Read, Write, and Control access by default for new files.

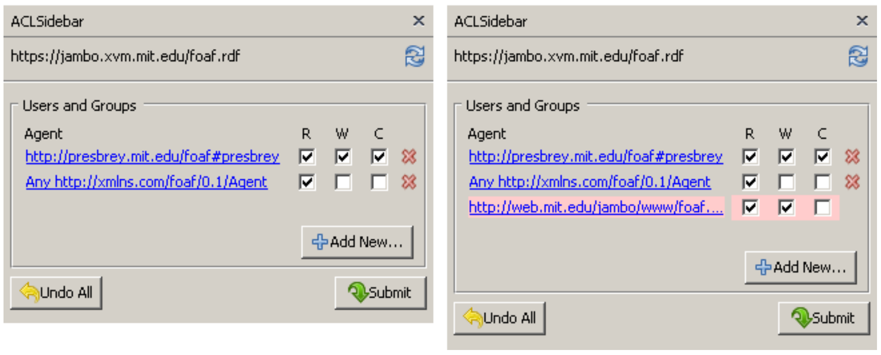
acl:mode	*HTTP Methods*
acl:Read	OPTIONS, GET, POST*, PROPFIND * interpretation of POST data must also indicate Read, e.g SELECT
acl:Write	PUT, DELETE, PROPPATCH, MKCOL, COPY*, MOVE** * two checks: Request URI acl:Read, Destination URI acl:Write ** two checks: Request URI acl:Write, Destination URI acl:Write
acl:Control	any acl:Write to an ACL URI

Table 1. Access control modes and their HTTP method mappings.

4 ACL Editor Interface

To demonstrate the functionality of the ACL editing system, we have developed a simple Firefox extension that allows users to edit the ACL for the document they are currently viewing. The extension is a sidebar that can be loaded along-side the current browser window. The ACL metadata for a given document is found by looking for the aforementioned `link rel=meta` HTTP header in the server's response for a given document. The metadata URI in the `link` header is dereferenced, and the ACL is displayed to the user as a list of agents and agent classes (See Figure 4).

Authorized users are able to modify the ACL directly in this interface by manipulating a series of checkboxes. New agents can be added, and current agents can be removed. When a user attempts to submit new ACL data, the interface serializes the state of the interface into a new ACL file. This modified file is then submitted to the server via a WebDAV PUT. Concurrent ACL modifications are prevented optimistically using the HTTP `ETag` header. Successful changes to the ACL take effect immediately.


ACLSidebar

https://jambo.xvm.mit.edu/foaf.rdf

Users and Groups

Agent	R	W	C
http://presbrey.mit.edu/foaf#presbrey	☑	☑	☑
Any http://xmlns.com/foaf/0.1/Agent	☑	☐	☐

ACLSidebar

https://jambo.xvm.mit.edu/foaf.rdf

Users and Groups

Agent	R	W	C
http://presbrey.mit.edu/foaf#presbrey	☑	☑	☑
Any http://xmlns.com/foaf/0.1/Agent	☑	☐	☐
http://web.mit.edu/jambo/www/foaf....	☑	☑	☐

Fig. 4. Viewing the ACL metadata from Figure 3 in the ACL Editor. On the left, the file is unmodified. On the right, the highlighted bottom row indicates the user has added the agent http://web.mit.edu/jambo/www/foaf.rdf#jambo, but not yet submitted the changes.

5 Performance

Though the FOAF+SSL authentication mechanism requires that the server make a request for a remote resource in order to perform authentication, it would be ideal for our system to nevertheless achieve performance comparable with traditional authentication mechanisms. To that end, we have performed performance tests in order to compare our authentication mechanism with traditional Apache authentication methods. The tests were run using ApacheBench, included with Apache, on an Apache instance running in single-process mode on a Quad-Core AMD Opteron 2350 2GHz Xen VM guest with 512MB of RAM. ApacheBench performed 1000 serial requests for a null document in each configuration. The four test configurations used tested the time needed for Apache to complete authentication requests depending on whether or not a certificate is presented and whether or not our authentication module is enabled. Table 2 shows the average performance of each configuration over the course of five test runs.

Apache Test Configuration	*Request Time (ms/request)*
without client certificate, without mod_authn_webid	4.2
without client certificate, with mod_authn_webid	4.43
with WebID client certificate, without mod_authn_webid	4.63
with WebID client certificate, with mod_authn_webid	11.31

Table 2. Benchmark results for mod_authn_webid.

The results clearly show that the module takes a few milliseconds to request the remote resource containing a user's certificate information. In this case, the request time is relatively short, since the two machines were sitting quite close to each other. With the addition of a WebID cache, though, the need for repeated requests for remote resources has been removed–a cached version of the agent's data is used instead for the duration of a session. In many cases, the inital request will not have very high latency, as in this example. But even in those cases where an initial request takes some time, subsequent requests will run at a speed comparable to normal certificate-based authentication, since the remote request is not repeated.

6 Future Work

With our first WebID- and Linked Data-based authorization system, we have created a highly functional codebase allowing for collaborative editing of Semantic Web documents. In this section, we describe a list of potential future enhancements that would greatly increase the ease with which users could edit data.

6.1 Packaging With SPARQL/Update-Ready Systems

Our current implementation uses WebDAV to allow for collaborative editing. However, this is by no means a requirement of the system. Over time, more systems are coming into existence that support SPARQL/Update for editing documents in an Apache environment, such as ARC2 [14] and the Tabulator Data Wiki [15]. These systems could be layered above our existing framework, with our framework limiting the HTTP methods a client can use (and, by extension, the sort of operations that a user can perform on a triple store). By switching over to SPARQL/Update, the size of updates sent by clients will be much smaller and have a lower potential for error, since clients will no longer be required to parse and serialize an entire document just to insert a single triple into the ACL data or triple store.

6.2 Per-Resource Access Control

While we currently provide access control at the document level, RDF is often stored as multiple resources described in a single flat file or in a triple store that does not even have a notion of specific documents. The current system is not able to restrict access on a per-resource basis unless resources are each stored in their own file. Such a system would require more complex analysis of an incoming update or query to determine exactly which resources are being read and written. While this feature is not included at present, it is worth noting that the ACL system used here could be modified to accommodate access defined on a per-resource basis. The first step in implementing such a system would probably be to have an engine capable of analyzing SPARQL/Update queries to decide which resources they affect.

Of course, one of the most difficult parts of adding per-resource access control is identifying exactly what it means for a read or write operation to "affect" a resource. A simple engine could designate a read or write operation as pertaining to a certain resource if the operation traverses or modifies an arc coming out of that resource. In this case, adding a "has_child" property to a parent would be considered a write modification to the parent, but not its new child. Some examples, however, can have greater consequences for a given resource. For example, if a user has write access to a class definition C, but is not granted write access to one of C's subclasses, the user could still greatly alter (or even break) the semantics of that subclass by simply modifying C. The importance of this issue will vary between use cases–for someone maintaining an open social network, the first method may be more appropriate; but for someone editing ontologies, maintaining consistency will be key.

6.3 Rule-based Access Control

This project uses a very simple form of access control–a simple whitelist contained in a metadata file. However, some users might want to define dynamic rules like the "friend-of-a-friend" rule described in the introduction. While such rules are certainly more descriptive than a whitelist, designing an interface that allows users to quickly modify rules and add new agents as our ACL Editor does is much more difficult. On the back-end, however, one would simply need to hook into a rule engine immediately prior to authorizing a user. At this point, our system exposes a user's URI and the query that the user is trying to run. This information could easily be plugged into an external reasoner to decide if the user is permitted to perform a given action or log the user's actions for later analysis. It should also be noted that using complex rules to regulate access to resources in a triple store could greatly affect the speed of the authentication process, since a proof would need to be constructed instead of searching for the appropriate triple indicating access.

7 Conclusion

The system described provides a solid base for future projects implementing Web Access Control and collaborative authoring of linked data. It provides an implementation of an RDF-based decentralized authentication protocol, namely FOAF+SSL. Authentication speeds, keeping in mind the need for an initial remote request, are comparable with traditional authentication systems. Users can quickly and easily add access for new agents without having to manually edit a metadata file.

That such a system can be implemented on a traditional Web server like Apache shows that there is potential for widespread adoption of accountable collaborative editing of linked data. The layout of this system allows for applications, linked data, and user agent authentication to all be stored on separate servers maintained by completely independent entities. By plugging services such as policy-aware query processors and data access logs into our system, one could create a complete policy-aware system based in an entirely decentralized environment, realizing the full potential of the Semantic Web.

Acknowledgments Thanks go to the members and students of DIG and the W3C who provided excellent knowledge and feedback throughout the course of this project. This work was supported by IARPA Award #FA8750-07-2-0031 and National Science Foundation Cybertrust Grant award CT-M: 0831442.

References

1. RFC 4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV), `http://tools.ietf.org/html/rfc4918`
2. SPARQL Update, `http://www.w3.org/Submission/SPARQL-Update/`
3. W3C ACL System, `http://www.w3.org/2001/04/20-ACLs`
4. The Apache Software Foundation, `http://www.apache.org/`
5. Recordon, D., Reed, D.: OpenID 2.0: a platform for user-centric identity management. In: Proceedings of the Second ACM Workshop on Digital Identity Management, pp. 11–16. ACM, New York (2006)
6. Story, H., Harbulot, B., Jacobi, I., Jones, M.: FOAF+SSL: RESTful Authentication for the Social Web. In: European Semantic Web Conference, Workshop: SPOT2009. Heraklion, Greece (2009)
7. FOAF Vocabulary Specification, `http://xmlns.com/foaf/spec/`
8. mod_ssl: The Apache Interface to OpenSSL, `http://www.modssl.org/`
9. Redland RDF Libraries, `http://librdf.org/`
10. SPARQL Query Language for RDF, `http://www.w3.org/TR/rdf-sparql-query/`
11. Notation3 (N3) A Readable RDF Syntax, `http://www.w3.org/DesignIssues/Notation3`
12. An Entity Header for Linked Resources, `http://www.w3.org/Protocols/9707-link-header.html`
13. Basic Access Control Ontology, `http://www.w3.org/ns/auth/acl#`
14. Easy RDF and SPARQL for LAMP Systems, `http://arc.semsol.org/`
15. Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., Prud'hommeaux, E., schraefel, mc.: Tabulator Redux: Browsing and writing linked data. In: WWW 2008, Workshop: Linked Data on the Web (LDOW 2008) (2008).

Appendix: Source Code

The source code for this project is available at the following locations:
Authentication Module: `http://dig.csail.mit.edu/2009/mod_authn_webid/`
Authorization Module: `http://dig.csail.mit.edu/2009/mod_authz_webid/`
ACL Editor: `http://dig.csail.mit.edu/2009/aclsidebar/`