The WebID Protocol Enhanced with Group Access Control

Maya Earl
Computer Science Department
North Carolina A&T State University
Greensboro, North Carolina
781-605-8621
mnearl@aggies.ncat.edu

Cory Sabol
Computer Science Department
University of North Carolina at Greensboro
Greensboro, North Carolina
336-618-2943
cssabol@uncg.edu

ABSTRACT
Through the use of various standards, such as, WebIDs and Node.js, we are improving the way entities interact with the web and authenticate their identities. WebID solves the challenge of remembering usernames and passwords, but a full realization of it will require the ability to manage groups of agents and control their access to resources on the Web. This paper explores the challenges and proposes a solution to the problem of delegating access to groups on the Web. Faculty advisor is Albert Esterline, esterlin@ncat.edu.

CCS Concepts
• Security and privacy~Access control • *Information systems~Resource Description Framework (RDF)*

Keywords
WebID; Group access; ACL; PKI; FOAF

1. INTRODUCTION
The web is vast and fragmented. The use of user name and password combinations to access resources on the web is a cumbersome and dated way of granting access and validating identity on the web. There is a desire to see the use of user names and password largely mitigated on the web. That is where alternatives such as WebID come into play. Instead of having to remember a user name and potentially complex password, a user can simply present a service with their WebID similarly to how one presents one's ID in the real world, and the service can then verify the user's identity and grant or deny access appropriately.

In this paper, we explore the implementation and design of a system that allocates resource access to groups of agents via WebIDs. As we shall see, there are some subtleties and challenges that arise in trying to do such a thing. In the remainder of this paper, we start with background, covering the technologies used, and then we describe the problem at hand. Next, we move to our solution and then to how we implemented it. Finally, we address where we plan to go in the future. This is a longer version and an enhancement of an earlier paper [3].

2. BACKGROUND
The technologies used include RDF, Node.js, FOAF, WebID, OpenSSL/TLS, ACL, and Linked-data.

2.1 RDF
The Resource Description Framework (RDF) is a specification for managing metadata information of resources on the web. We will refer to this framework as RDF moving forward. It essentially extends the structure of linked data via URIs (Universal Resource Identifier). The RDF model expresses resources as subject, predicate and object; this relationship is known as 'triples' [7]. With this model, we can build a framework of relationships between agents and the documents they control. The *subject* is the actual resource. The *predicate* indicates which characteristic of the resource is being referenced, i.e. the relationship between the *object* and the *subject*. The *object* is the value of the specified characteristic.

RDF has the versatility to be expressed in XML, for machine understanding, or N3, for a more human readable format. The Resource Description Framework Schema (RDFS) allows us to build RDF vocabularies such as FOAF or ACL to enable efficient integration and interoperability of data. Both vocabularies will be described in the following sub-sections.

2.2 Node.js
Node.js is a server-side implementation of JavaScript built on Google's V8 JavaScript engine [1]. Node applications are single threaded and event driven, allowing us easily to write asynchronous or non-blocking code, which is very important for developing web based applications.

We use Node.js to implement all of our code and the WebID protocol. It affords minimal hassle in terms of set up and lets us rapidly prototype and test with frameworks such as Mocha.

2.3 FOAF
Friend of a Friend, or FOAF, is an RDF schema used to describe people and their social networks in a semantic way [9]. It is applicable to more than just people, however, and can be easily applied to agents on the web in general. The conventional namespace prefix for FOAF terms is **`foaf`**.

Of particular importance is the term **`foaf:Group`**. It defines an agent as a group of members that nevertheless can be treated as an individual. This is critical in allowing a group to identify its members. Note that the property **`foaf:membershipClass`** associates an **`owl:Class`** with a **`foaf:Group`**: a member of the group automatically has the class as its type and vice versa.

2.4 WebID
WebID is a way to identify an agent using web technologies [6]. A WebID is a unique URI that dereferences to an RDF document with a profile of the denoted agent. (Dereferencing is getting the document pointed to by the URI.) This uses CoolURIs [8], which dereference to the appropriate document on what they identify using content-negotiation along with 303 redirects or fragment identifiers. The service chooses what kind of document to provide, typically RDF to a machine agent and HTML to a human. In the profile document, the agent's public key is published so that, when the agent presents its certificate, the public key in it can be verified against that in the document.

PKI (public key infrastructure) also plays a large role in the implementation of WebID. PKI is a set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption. In the profile document that a WebID dereferences to, the agent's public key must be published. This is so that, when the agent presents its certificate, which also contains their public key, it can be compared with the public key in the document for verification.

2.5 OpenSSL/TLS

OpenSSL is an open source implementation of the SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols. It is also a general-purpose cryptography library [10]. It is used to implement the WebID protocol in a secure manner. We also utilize the OpenSSL library to generate client side x509 certificates, which are used by an agent to identify itself according to the WebID specification.

2.6 ACL

The Access Control Language (ACL) is used to delegate various forms of access to authenticate agents in a decentralized manner [11]. Specifically, in the group aspect of the WebID specification, we would want to allow access to a particular resource based on who is asking for permissions. **Table 1** shows the resource modes that ACL provides.

Table 1. ACL Resource Modes

Mode:	Those allowed may:
Read	read the contents (including querying it, etc)
Write	overwrite contents (including deleting or modifying part of it).
Append	add information to [the end of] it but not remove information.
Control	set the Access Control List for this themselves.

This ACLRDF vocabulary, which is defined with RDFS, includes properties of authorization [11]. It allows the administrator to specify access control modes, which are essentially classes of operation. The protocol states that the user authenticates to the server through the implementation of WebID+TLS, which we will cover in the following section.

2.7 Linked-data

Linked-data is a set of best practices regarding the publication and connection of structured data on the Web [2]. Linked-data allows for the web to have one consistent data source such that many different machine agents as well as human agents can all communicate with one another seamlessly and can discover and consume new data easily. This is all due to RDF, which is the crux of linked-data.

3. WEBID + TLS

The WebID-TLS protocol enables secure, efficient and maximally user-friendly authentication on the Web [12]. This is vital in propagating access only to users that have been identified and certified.

Protocol

To provide a full context of a client's interaction with the server, the protocol is defined by the following steps [12]:

1. Initially, the client must establish a TLS connection, which the server authenticates itself, using the standard TLS protocol.
2. Next, the client on a specific resource can perform an HTTP GET, PUT, POST, or DELETE action.
3. At this stage, the guard, or server agent,can grant or deny access according to the access control rules.
4. Now, if the resource requires authentication for access, the client's identity is linked to a private key and public key pair. This pair should be embedded in the client's certificate to be verified. The client has the option to automate which certificate to send from its browser when the TLS agent makes a Certificate Request. The TLS agent matches the certificate public key with the public key in the user's profile document on the server.
5. The Verification Agent, defined in [12], verifies that the WebID in the WebID certificate knows the given public key.
6. The Verification Agent extracts the public key and the URI from the *Subject Alternative Name* property field in the certificate.
7. With the WebID collected, the guard can check if one of them is authorized by the access control rules.
8. If access is granted, then the guard can pass on the request to the protected resource.

4. THE PROBLEM

The current ways of handling security and control of access to resources on the web are bulky and depend on static data and implementations local to the services themselves. The advent of linked-data and WebIDs provides the means to implement a more distributed and interconnected way of delegating access to resources on the web.

This problem is addressed using WAC (WebAccessControl), which is a specification for implementing access control using linked-data technologies. It relies heavily on ACL. A more subtle problem, however, is that of group access control. We describe a way in which this problem can be solved and discuss some of the nuances of the **`foaf:Group`** class.

There is, at the time of writing, another group of researchers, who form the linkeddata [4] group, and they have had some success in implementing access control for groups and other things, such as WebID verification using Node.js, which is used in our implementation. We have collaborated and been in contact with some of the members regarding WebID provisioning using Node.js.

Groups can arise in a natural way using linked-data. One could model all of the relationships of an agent's social network as a digraph or graph and treat that as the group. However, this is a more or less inferred group, which does not work well with the web and WAC. This is just another part of the larger problem of group access control.

5. SOLUTION

We must make sure a group is explicitly defined to help in dealing with groups of agents defined with linked-data. This is where the **`foaf:Group`** class comes in. Figure 1 shows a simple example profile document of a group agent. The N3 serialization of RDF is used for readability. The profile asserts the name of the group and contains an array of WebIDs that represent the members of the group, defined using the **`foaf:member`** term. The document also contains the agent's public key, used for the WebID verification process.

```
@prefix foaf: <http://www.xmlns.com/foaf/spec/#>.
@prefix cert: <http://www.w3.org/ns/auth/cert/#>.
```

```

_:webidcommunitygroup a foaf:Group;
        foaf:name "webidcommunitygroup";
        foaf:member[/*group member WebID's here */];
        cert:PublicKey "MIICQ…EstKg==" .
```

Figure 1. Example Profile Document (N3)

We can allow a WebID that identifies a group to serve as the entry point to allotting access to group members. In addition, the group itself can act as an individual, making assertions for its group members. It follows that group members should also identify their membership to the group within their profile document.

Figure 2 presents a general algorithm for propagating access to the members of a group. On lines 1 through 18, we define the function **`grantAccess`**, which takes in a URI representing the agent requesting access and a URI representing the resource in question. On line 2, we immediately check to make sure the agent's WebID was successfully validated; if not, then we deny access and alert the resource owner. On line 7, we dereference the agent's URI to get their profile document, and then, on line 9, we parse the document into a graph. We do the same for the resource and, getting its ACL file, we parse it to a graph on lines 8 and 10. In line 12, we then check to see whether the agent is explicitly mentioned in the resource ACL (**`ResACLGraph`**) or the agent's class is explicitly mentioned in the resource ACL. If so, we grant the access specified by the ACL (line 13). Otherwise, access is denied (lines 16-17).

```
1: define function grantAccess(AgentURI, ResourceURI):
2:   if (Agent's WebID not valid) then
3:      Deny access to ResourceURI, prompt resource owner
4:      that Agent has requested access to the resource.
5:      return access denied
5:   end
6:
7:   Define AProfile := dereference(AgentURI)
8:   Define ResACL := dereference(ResourceURI)
9:   Define AGraph := toGraph(AProfile)
10:  Define ResACLGraph := toGraph(ResACL)
11:
12:  if (AgentURI is in ResACLGraph OR
         AGraph.class is in ResACLGraph) then
13:     return access type specified in ResACL
14:  end
15:  else
16:     Deny access to ResourceURI, prompt resource owner
17:     that Agent has requested access to the resource.
18:     return access type specified in ResACL
18: end
19:
20: forEach (Member m in Group g) do
21:   accessCache.add(m, grantAccess(m, Resource.URI))
22: end
```

Figure 2. General Access Algorithm

In lines 20-22, we iterate through each member listed in the Group **`g`** and, for each member **`m`**, we call our function **`grantAccess`**, passing it the **`m`**'s URI and the URI of the resource. We then pass the function's return value to the **`accessCache.add()`** method. Implementation of **`grantAccess()`** is up to the developer, but the information it returns must be useful enough to be cached and used by the service to make access decisions in the future.

6. IMPLEMENTATION

Node.js is the implementation language. We have implemented WebID for the first layer of authentication. Following WebAccessControl, our code denies or allows access to server endpoints based on the WebID and the endpoint's ACL rules. We use the Express [5] Node.js framework for request handling. When a user connects to a service, they are asked for their certificate. The certificate is verified by the service, and the user is directed to either the access-granted or an access-denied page. Use of certificate verification allows for a shorter process of determining who has access. The code takes in the path to an RDF file with ACL content, which is passed to the reasoner. The code then returns a response that directs the code to either allow or deny access.

We determine whether the authenticated agent is a group by checking its profile for **`foaf:Group`**. If it is a group, the algorithm in Figure 2 is used. The code either returns the appropriate representation of the resource or redirects to an access-denied page, and the attempted access is logged. The code treats a group differently from a non-group agent only in that it authenticates and stores access data for the group's members. Otherwise, a group can act and be treated as an individual just like any non-group agent.

7. FUTURE WORK

Going forward, we must continue to refine the ideas presented in this paper and our implementation of them. We will test various access cases by designing several tests, using the Mocha and Chai frameworks mentioned earlier. The method of caching the access data to the service will be extended and refined and integrated into the algorithm presented here. We will also explore issues that arise from group access to resources, including issues of trust in a social network. We will model trust in a linked data social network and implement techniques to infer access control based on a trust metric. There are several long-term goals we wish to achieve, including the following:

- Incorporate biometric work to have other ways to determine someone's identity, specifically, through their actions or physical features.
- Have the service that provides the WebID be a complete agent.
- Allow agent-based, automated collaboration between users' systems. This would provide features of multiagent systems and of grid computing.
- Incorporate a policy ontology and a reasoner to allow us to enforce access policies.

8. CONCLUSION

Users of the web have always built on the web to make information more accessible yet secure and faster to access. Automating the relationship between the server and client to ensure agents are who they say they are allows the web to accomplish these goals. Achieving these goals required in depth research into the typical Semantic Web technology stack. RDF, ACL, FOAF, and TLS are among the most useful standards in our work. We have described the design and implementation of a system that allocates resource access to groups of agents via WebIDs.

9. ACKNOWLEDGEMENT

This research is based upon work supported by the U.S. Government, including the NSF REU (grant #1460864) and the ARO (Contract No. W911NF-15-1-0524). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. We would like to express gratitude to ADMI for their time and consideration.

10. REFERENCES

[1] "About Node.js." About. from https://nodejs.org/en/about/.

[2] C. Bizer, T. Heath, and T. Berners-Lee (2009). "Linked Data—The Story So Far." Int. J. on Semantic Web and Info. Systems pp. 1–22."OpenSSL." *OpenSSL*. N.p., n.d. Web. 26 Dec. 2015. <https://www.openssl.org/>.

[3] C. Sabol, W. Odd. "Group Access Control Using WebID." Submitted to IEEE South East Conference 2016.

[4] D. Ayers and M. Völkel. (2008). "Cool URIs for the Semantic Web." Cool URIs for the Semantic Web. from http://www.w3.org/TR/cooluris/.

[5] "Express - Node.js Web Application Framework." Express - Node.js Web Application Framework. from http://expressjs.com.

[6] H. Story, T. Inkster, and B. Harbulot. (2014). "WebID-TLS " W3C.

[7] J. J. C. Graham Klyne, B. McBride. (2014). "RDF Concepts and Abstract Syntax." from http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[8] "Linkeddata." Github. from https://github.com/linkeddata.

[9] L. M. D. Brickley. (2014). "FOAF Vocabulary Specification 0.99 (Paddington ed.)." from http://xmlns.com/foaf/spec/.

[10] "OpenSSL." OpenSSL. from https://www.openssl.org/.

[11] "WebAccessControl." WebAccessControl. from http://www.w3.org/wiki/WebAccessControl.

[12] W3C, "WebID-TLS," in Authentification over TLS, ed: W3C, 2015.