DASI Breaker: WAC
Report

Simone Persiani, Ludovico Granata | 13/05/2021

AUTHORIZATION LAYER

Current SEPA implementation exploits the graph level access control offered by Virtuoso Open Source through the OpenID Connect interface provided by Keycloak.

This task aims to extend this mechanism to make it independent from the underline SPARQL endpoint via the Web Access Control (WAC) specification.

Overview

The Web Access Control (WAC) layer is responsible for authorizing agents (users, groups and more) to perform various kinds of operations (read, write, append, etc) on LDP Resources.

WAC has several key features:

- the protected resources are identified by URLs;
- Access-Control policies are persisted as **a special kind of resources**, which can be exported/modified easily;
- users and groups are also identified by URLs (specifically, by **WebIDs**);
- ***cross-domain***: all components, such as resources, agent WebIDs, documents containing the Access-Control policies, can potentially reside on separate domains.

Source: Overview | Web Access Control (WAC) specification (as used by the Solid project)

Access Control List Resources

Each resource has a set of *Authorization* statements :

- Who has access to that resource (that is, who the authorized agents are)
- What types (or modes) of access they have

Authorizations are either explicitly set for an individual resource, or (more often) inherited from that resource's parent folder or container.

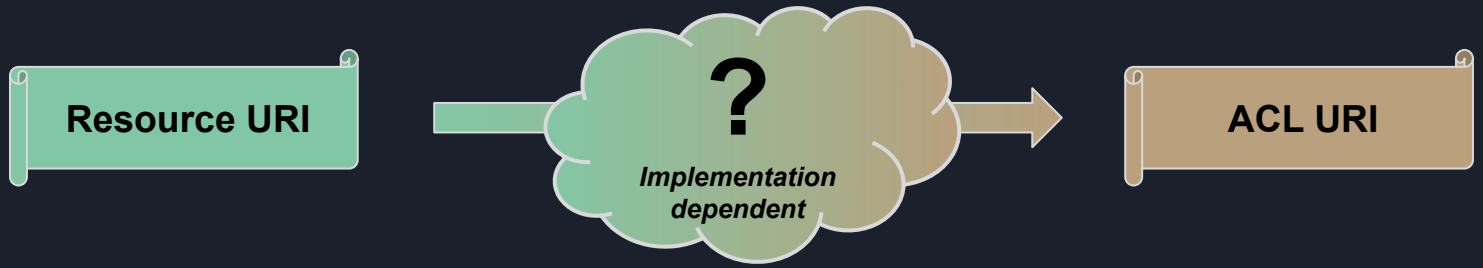
Authorization statements are placed into separate WAC documents called *Access Control List Resources* (or simply *ACLs*).

Source: Access Control List Resources | Web Access Control (WAC) specification (as used by the Solid project)

ACL Resource Location - 1

Given a URL for an individual resource or container, a client can discover the location of its corresponding ACL by performing a HEAD (or a GET) request and parsing the rel="acl" link relation.

Clients MUST NOT assume that the location of an ACL resource can be deterministically derived from a document's URL. The actual naming convention for ACL resources can differ for each individual implementation (or even for each individual server).



Source: ACL Resource Location Discovery | Web Access Control (WAC) specification (as used by the Solid project)

ACL Resource Location - 2

Request to discover the location of the ACL resource for a web document at `http://example.org/docs/file1`

```
HEAD /docs/file1 HTTP/1.1
Host: example.org

HTTP/1.1 200 OK
Link: <file1.acl>; rel="acl"
```

Request to discover the location of a container's ACL resource

```
HEAD /docs/ HTTP/1.1
Host: example.org

HTTP/1.1 200 OK
Link: <.acl>; rel="acl"
```

Source: ACL Resource Location Discovery | Web Access Control (WAC) specification (as used by the Solid project)

ACL Inheritance Algorithm

1. Use the document's own ACL resource if it exists (in which case, stop here).
2. Otherwise, look for authorizations to inherit from the ACL of the document's container. If those are found, stop here.
3. Failing that, check the container's parent container to see if that has its own ACL file, and see if there are any permissions to inherit.
4. Failing that, move up the container hierarchy until you find a container with an existing ACL file, which has some permissions to inherit.
5. The root container of a user's account MUST have an ACL resource specified. (If all else fails, the search stops there.)

Source: ACL Inheritance Algorithm | Web Access Control (WAC) specification (as used by the Solid project)

ACL Resources and LDP

Even though the ACL file is neither an LDP Resource nor an LDP Container, it should be editable using the same methods.

- ACLs can be edited using the same verbs as those used by LDP: PUT and PATCH.
- Only agents with the acl:Control privilege can edit the ACL file itself
- It is arguable that an ACL file should not be DELETEable, as it is created by the server on creation of the resource. On the other hand, deleting the resource should also delete the associated ACL.

Source: WAC and LDP | WebAccessControl - W3C Wiki

ACL Ontology - 1

ACL ontology: https://www.w3.org/ns/auth/acl

Access modes:

- **acl:Read**
- **acl:Write**
- **acl:Append** - A typical example of Append mode usage would be a user's Inbox -- other agents can write (append) notifications to the inbox, but cannot alter or read existing ones.
- **acl:Control** - is a special-case access mode that gives an agent the ability to view and modify the ACL of a resource. For example, a resource owner may disable their own Write access (to prevent accidental over-writing of a resource by an app), but be able to change their access levels at a later point (since they retain acl:Control access).

Source: Modes of Access | Web Access Control (WAC) specification (as used by the Solid project)

ACL Ontology - 2

Privileges for a Singular Agent:

```
# Contents of https://alice.databox.me/docs/file1.acl
@prefix  acl:  <http://www.w3.org/ns/auth/acl#>  .

<#authorization1>
    a             acl:Authorization;
    acl:agent     <https://alice.databox.me/profile/card#me>;  # Alice's WebID
    acl:accessTo  <https://alice.databox.me/docs/file1>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control.
```

Source: Singular Agent | Web Access Control (WAC) specification (as used by the Solid project)

ACL Ontology - 3

Privileges for Groups of Agents:

```
# Contents of https://alice.databox.me/docs/shared-file1.acl
@prefix  acl:  <http://www.w3.org/ns/auth/acl#>.

# Individual authorization - Alice has Read/Write/Control access
<#authorization1>
    a             acl:Authorization;
    acl:accessTo  <https://alice.example.com/docs/shared-file1>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control;
    acl:agent     <https://alice.example.com/profile/card#me>.

# Group authorization, giving Read/Write access to two groups, which are
# specified in the 'work-groups' document.
<#authorization2>
    a              acl:Authorization;
    acl:accessTo   <https://alice.example.com/docs/shared-file1>;
    acl:mode       acl:Read,
                   acl:Write;
    acl:agentGroup <https://alice.example.com/work-groups#Accounting>;
    acl:agentGroup <https://alice.example.com/work-groups#Management>.
```

Group file:

```
# Contents of https://alice.example.com/work-groups
@prefix    acl:  <http://www.w3.org/ns/auth/acl#>.
@prefix     dc:  <http://purl.org/dc/terms/>.
@prefix  vcard:  <http://www.w3.org/2006/vcard/ns#>.
@prefix    xsd:  <http://www.w3.org/2001/XMLSchema#>.

<#Accounting>
    a               vcard:Group;
    vcard:hasUID    <urn:uuid:8831CBAD-1111-2222-8563-F0F4787E5398:ABGroup>;
    dc:created      "2013-09-11T07:18:19Z"^^xsd:dateTime;
    dc:modified     "2015-08-08T14:45:15Z"^^xsd:dateTime;

    # Accounting group members:
    vcard:hasMember  <https://bob.example.com/profile/card#me>;
    vcard:hasMember  <https://candice.example.com/profile/card#me>.

<#Management>
    a               vcard:Group;
    vcard:hasUID    <urn:uuid:8831CBAD-3333-4444-8563-F0F4787E5398:ABGroup>;

    # Management group members:
    vcard:hasMember  <https://deb.example.com/profile/card#me>.
```

Source: Groups of Agents | Web Access Control (WAC) specification (as used by the Solid project)

ACL Ontology - 4

Privileges for Public Access (All Agents):

```
@prefix   acl:  <http://www.w3.org/ns/auth/acl#>.
@prefix  foaf:  <http://xmlns.com/foaf/0.1/>.

<#authorization2>
    a               acl:Authorization;
    acl:agentClass  foaf:Agent;                    # everyone
    acl:mode        acl:Read;                      # has Read-only access
    acl:accessTo    <https://alice.databox.me/profile/card>.  # to the public profile
```

Source: Public Access (All Agents) | Web Access Control (WAC) specification (as used by the Solid project)

ACL Ontology - 5

Privileges for Authenticated Agents (Anyone logged on):

```
@prefix   acl:  <http://www.w3.org/ns/auth/acl#>.
@prefix  foaf:  <http://xmlns.com/foaf/0.1/>.

<#authorization2>
    a               acl:Authorization;
    acl:agentClass  acl:AuthenticatedAgent;          # everyone
    acl:mode        acl:Read;                        # has Read-only access
    acl:accessTo    <https://alice.databox.me/profile/card>.  # to the public profile
```

Note that this is a special case of acl:agentClass usage, since it doesn't point to a Class Listing document that's meant to be de-referenced.

Source: Authenticated Agents (Anyone logged on) | Web Access Control (WAC) specification (as used by the Solid project)

AUTHENTICATION LAYER

Current SEPA implementation exploits the graph level access control offered by Virtuoso Open Source through the OpenID Connect interface provided by Keycloak.

This task aims to extend this mechanism to make it independent from the underline SPARQL endpoint via the Web Access Control (WAC) specification.

WebID

- Agents are identified through their WebIDs

  A WebID is a way to uniquely identify a person, company, organisation, or other agent using a URI.

- How can we make a WebID?
  1. We host a profile document (FOAF file) on a public domain
  2. The WebID URI will be the concatenation between the URL of that file on the Web and a fragment (#YOUR_INITIALS, #me or #this)

EXAMPLE:

http://your.isp.com/whatever/~yourusername/foaf.rdf#ABC

Source: Can I make my own Web ID? | WebID - W3C Wiki

WebID - Available protocols

There are two main protocols to provide authentication through WebIDs:

- WebID-TLS

  Protocol that allows a service to authenticate a user without needing to rely on this being signed by a well known Certificate Authority. To do this we have to generate a new certificate and then publish the public key inside the FOAF file (our profile): in this way we will be able to verify our identity by using the private key during a TLS handshake.

- WebID-OIDC

  Protocol based on OAuth2/OpenID Connect. The WebID-OIDC protocol specifies a mechanism for getting a WebID URI from an OIDC ID Token, inheriting the benefits of both the decentralized flexibility of WebID and of the field-proven security of OpenID Connect.

WebID-TLS

The user can signal the existence of its Web account by storing a WebID profile (an RDF text file) on a publicly available web server.

A personal web server can be used for this purpose or, as an alternative, online services do exist that allow anyone to create and host their WebID profile.

In order to be able to authenticate using the WebID-TLS protocol, the user must generate a new certificate and then add to its WebID profile some triples containing the public key.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<#JW>
    a foaf:Person ;
    foaf:name "James Wales" ;
    foaf:mbox <mailto:jwales@bomis.com> ;
    foaf:homepage <http://www.jameswales.com> ;
    foaf:nick "Jimbo" ;
    foaf:depiction <http://www.jameswales.com/aus_img_small.jpg> ;
    foaf:interest <http://www.wikimedia.org> ;
    foaf:knows [
        a foaf:Person ;
        foaf:name "Angela Beesley"
    ] .

<http://www.wikimedia.org>
    rdfs:label "Wikimedia" .
```

```
@prefix cert:  <http://www.w3.org/ns/auth/cert#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .


<myfoaf#me> cert:key [
    cert:exponent 65537;
    cert:modulus "cb24ed85d64d79 [...] c61391a1"^^xsd:hexBinary;
] .
```

Source: Adding a certificate | WebID - W3C Wiki

WebID-OIDC

An OIDC-compliant Identity Provider server is required, such as Keycloak.

Authentication is done by the IDP via the user's preferred method (password, 2FA, QR code, etc). Upon a successful authentication, the IDP generates a signed ID Token with a webid claim containing the WebID URI of the user.

The IDP should internally store the WebIDs associated to the users, ensuring that a new user cannot choose an already registered WebID. New WebIDs can be chosen by the user or automatically generated: in either case, they must share the same domain.

EXAMPLE
http://webid.service-domain.org/<unique path/identifier>#me

Together with the new WebID URI, a WebID profile containing user information should be created and made publicly accessible by browsing to the WebID URL address.

WebID-OIDC Workflow - 1

Let's assume that Alice tries to request the resource `https://bob.example/resource1`

1) Initial Request	2) Provider Selection	3) Local Authentication	4) User consent (optional)
Alice (unauthenticated) makes a request to bob.example. She receives an HTTP 401 Unauthorized response, and is presented with a 'Sign In With...' screen.	She selects her WebID service provider by clicking on a logo, typing in a URI (for example, alice.solidtest.space), or entering her email.	Alice gets redirected towards her service provider's own Sign In page, thus requesting https://alice.solidtest.space/signin, and authenticates using her preferred method (password, WebID-TLS certificate, FIDO 2 / WebAuthn device, etc).	She'd also be presented with a user consent screen, along the lines of "Do you wish to sign in to bob.example?".

Source: Brief Workflow Summary | Specs for WebID-OIDC decentralized authentication protocol (based on OAuth2/OpenID Connect)

WebID-OIDC Workflow - 2

Let's assume that Alice tries to request the resource `https://bob.example/resource1`

5) Authentication Response

She then gets redirected back towards https://bob.example/resource1 (the resource she was originally trying to request). The server, bob.example, also receives a signed ID Token from alice.solidtest.space that was returned with the response in point 3, attesting that she has signed in.

6) Deriving a WebID URI

bob.example (the server controlling the resource) validates the ID Token, and extracts Alice's WebID URI from inside it. She is now signed in to bob.example as user https://alice.solidtest.space/#i.

7) WebID Provider Confirmation

bob.example confirms that solidtest.space is indeed Alice's authorized OIDC provider (by matching the provider URI from the iss claim with Alice's WebID).

Source: Brief Workflow Summary | Specs for WebID-OIDC decentralized authentication protocol (based on OAuth2/OpenID Connect)

State-of-the-art implementations

Community Solid Server

An open and modular implementation of the Solid specifications :
“its modular architecture allows trying out new ideas on the server side and thereby shape the future of Solid.”

Implements:

- WAC
- LDP
- Pod creation is not yet supported fully
- we must rely on an external identity provider to log in and authenticate our WebID

Architecture

Source: Community Solid Server: an open and modular implementation of the Solid specifications

DEMO

Community Solid Server + Blazegraph