Design - WebAccessControl Authorization Delegate

This page will be used to design a WebAccessControl Authorization Delegate.

Use Cases

Phase 1

1. As a user from the Registrar office creates an Asset with a loan agreement document, the user assigns a property to the asset indicating that the asset is restricted to the Registrar staff, the user (a member of the Registrar group) should not be locked out of viewing/editing the resource.
2. A user wishes to create multiple resources that all share access restrictions. The user is able to create a single access control list that is used by that group of resources. In addition, they do not have to explicitly link to the ACL from every resource; the appropriate ACL is inferred from the collection a resource is a member of (collection defined as a resource that ldp:contains the target resource) or the class(es) it is an instance of.
3. There is a collection that is by default restricted access. However, there are a selection of resources within that collection that are publicly available. These public resources are not necessarily grouped together in a single hierarchy within the collection.
4. An unauthenticated user requests a protected resource. The system presents them with an authentication challenge. The user successfully authenticates, and it is determined that they have read access to the requested resource. The system retrieves and presents the resource to the user.
    a. Some means of simulating this scenario will have to be devised for testing purposes, since presumably the test Vagrant will not be connected to an authentication service.
5. An authorized user must be able to change which ACL applies to a given resource or group of resources without moving those resources.
6. An authorized user must be able to edit an ACL in place in order to update the policy that applies to all resources governed by that ACL.
7. An ACL should be its own ACL; thus, a user who has read (or write) access to a given ACL should be able to read (or write) to that ACL. A user who has no access according to a given ACL should not be able to view the ACL.
8. A collection of books has an ACL that allows anonymous read access. One book within that collection (book A) has more restrictive access rules (i.e. only certain authorized users can read it), and these rules are defined in a different ACL that book A points to. Another book in the same collection (book B) points to no ACL directly, but should be governed by the ACL of the parent collection. When an anonymous user attempts to access book A, access should be denied (due to the ACL that book A points to), but when the same user tries to access book B, access is allowed by virtue of the ACL that the parent collection points to.
9. The opposite scenario to #8 above – i.e. where the collection has **more restrictive** rules and the child book has **more permissive** rules – should also be possible by simply reversing the two ACLs.

Phase 2 - proposed

1. A resource can be related to tags. Each tag is contained in a tag category: Conservation, Imaging, Licensing, etc. For tags related to a certain resource:
    a. Imaging users can read, create and delete tags in Imaging category
    b. Imaging users can read tags in Conservation category
    c. Conservation users can read, create and delete tags in Conservation and Imaging categories
    - *Note: F4 does not have "tags" per se. Are you envisioning a "tag" as a resource? or a property?*
    - *See comment below*
2. Enforce ACLs on binary resources
3. Add support for external ACL resources (resources that point to an ACL that is outside the Fedora domain)
4. Add support for external agentClass graphs
5. Enforce ACLs on ACL resources with a filesystem-based backstop (at present, only the admin user can add/edit ACLs)
6. Verify header-based (delegated) authentication use cases
7. Add ACL URIs to response headers as `Link: <acl-uri>; rel=meta`
8. Implement `acl:Control` and `acl:Append` modes

9. Add support for inclusion of other acls via `acl:include`

Removed/Amended Use Cases

Amended

1. Phase 1, #2 - A user wishes to create multiple resources that all share access restrictions. The user is able to create a single access control list that is used by that group of resources. In addition, they do not have to explicitly link to the ACL from every resource; the appropriate ACL is inferred from the collection a resource is a member of, the class(es) is is an instance of, or other metadata associated with the resource.
    a. Amended to remove the underlined context above and to define "*is a member of*" as a resource which ldp:contains the target.

Removed

1. An unauthenticated user requests a publicly readable resource. The system retrieves and presents the resource to the user without any authentication challenge.
    a. See this comment below
2. It should be possible to create an ACL that changes over time without user intervention (e.g. an embargo policy).
    a. This type of operations should make use of the proposed API extension architecture and a front-end web service to handle expiration of policies and leave Fedora to only apply the policies at the time of the request.

Questions arising from use cases

1. Given these multiple ways of inferring which ACLs apply to a given resource (via collection, class, or other metadata), there must be clear rules for determining the precedence of policies and for resolving conflicts between policies.
    a. See Finding the effective policy set below.
2. In the interest of security, the absence of an applicable ACL should result in denial of Create, Update, and Delete requests (and possibly also Read requests?) to non-admin users, but should allow all CRUD operations to admin users.
    a. the absence of an applicable ACL will result in denial of all requests.

Proposed Requirements (Phase 1)

Legend

- ✅ Completed
- ❌ Not yet completed

Sprint 1 (Phase I) Requirements

1. ✅ F4 MUST allow assertions about authorization to be modeled in RDF in accordance with the WebAccessControl specification.
    a. Access assertions to be implemented are
        i. ✅ READ -> GET a resource
        ii. ✅ WRITE -> PUT/POST/PATCH/DELETE a resource
        iii. ❌ APPEND -> PATCH a resource, restricted to Insert statements only.
    b. as well as extending for:
        i. ❌ DELETE -> DELETE a resource
        ii. ❌ UPDATE -> PUT/POST a resource
    c. ❌ The implementation assumes an ACL is it's own ACL, therefore CONTROL will not be implemented at this time.
    d. Optional extensions (ie. regex matching) will not be implemented in Phase 1.
2. ✅ F4 MUST be able to enforce authorization based on WebAC when a resource is requested via the REST-API
3. ✅ F4 MUST allow authorization policies to apply to a group of resources which consists of:
    a. ✅ Resources sharing a rdf:type attribute matching an acl:accessToClass rule in an ACL in the preconfigured location.
        i. *Note: sprint-1 implementation does not confine ACLs to reside in a "preconfigured location", but they can instead exist anywhere within the repository.*
    b. ✅ Resources (without their own specific policy) share the policy of their container (defined as the resource which ldp:contains the target).
4. ✅ F4 MUST honor the most permissive authorization policy when multiple policies apply to a request.
    a. See the section Finding the effective policy set, for more clarity.
5. ❌ F4 MUST provide a way for external services such as Solr to enforce the authorization rules defined in the repository.
    a. Would one method of achieving this be to create separate solr cores for public users vs. admin users and publish only the public resources to the former and everything to the latter, or does the use case dictate more granular distinctions?
6. ❌ Each request for a Web Resource returns an HTTP document containing a Link header (Link: <>; rel=acl) to an ACL resource which describes access to the given resource and potentially others.
7. ✅ Servers are required to recognize the class foaf:Agent as the class of all agents (i.e. foaf:Agent is synonymous with "everyone")

Sprint 2 (Phase I) Requirements

Must have

1. Enforce A
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
2. Implemen
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
3. F4 MUST
   a. and
   b. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
4. Enforce A
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
5. More doc
   a.
   b.
   c. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
6. Add supp
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
7. Verify he ndependent of any container-based Au
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
8. Fix bug w
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
9. Make we
   a.
   b. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
10. Bug fixes
    a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Nice to have

1. Add ACL
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.
2. Support e
3. Add supp
4. Minor co
   a. Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Likely not

1. Support for inclusion of other ACLs via acl:include

Removed requirements

1. F4 resources that are open for public read should not challenge the client to authenticate
   a. What if a resource has no ACL? Should there be a default behavior? (for example allow all to admin user; deny all to non-admin?)
   b. This requirement specifies public *read*. Do we also want to allow a public write? While use cases for the latter are admittedly going to be rare, it would seem better not to be opinionated here if public write is what an implementer wants.

See this comment below for more information on this removal.

Finding the effective policy set

Finding the effective ACL

1. Use ACL that directly references target resource, if exists, else
   a. if multiple ACLs apply to a given target resource, the most permissive is used.
2. Use ACL from configured location that has policy for target resource class, if exists, else
   a. if multiple ACLs apply to a given class, the most permissive is used.
   b. *accessToClass* statements in ACLs **not** in the configured location are ignored.
3. Recursively follow steps 1 and 2 for parent resource that ldp:contains target resource, if exists, else
4. Deny access

Resolving the effective policy from an ACL

1. Use policy that specifies requesting **userId**, if exists, else
2. Use policy that specifies requesting **groupId**, if exists, else
    a. Note, if multiple requesting groupIds have policies, use the one that grants the most access.
3. Deny access

Authentication Considerations

[NOTE: This section has been stricken because it is not germane to the specific effort to develop a WebACL authorization delegate; the authentication considerations described below need to be part of the larger configuration of the ways Fedora and the web server interact, but that is a separate issue.]

Fedora 4 always assumes that any incoming request has always been authenticated by the container (or other layers above it in the stack). Therefore, these considerations may not be central to the design of a new authorization delegate; however, I (peichman) do believe they represent an important part of setting up a Fedora instance (beyond just the core servlet code) and should be addressed in some form.

1. Authorization logic should be capable of evaluating four pieces of information about a given request: (1) Origin (IP address), (2) identity of requester (or anonymous if not authenticated), (3) the resource being requested, and (4) the method or action of the request (read/write, GET /POST, etc.).
2. After evaluating the WebACL policies that apply to the requested resource against the four pieces of information listed above, the application should return one of three responses: (1) fulfill the request, (2) prompt the user to authenticate, or (3) deny the request.

Open Questions

1. Is there anything currently implemented within the Hydra WebAC implementation that strays from direct compatibility with the WebAC standard? In other words, are there currently barriers to the goal of cross-application compatibility?

Role Commitments

Development

- Peter Eichman
- Mohamed Mohideen Abdul Rasheed
- Jared Whiklo
- Unknown User (acoburn)

Stakeholder

- Joshua Westgard
- Stefano Cossu
- Nick Ruest

Related Documents

- https://www.w3.org/wiki/WebAccessControl (note that this is still a wiki, not yet a draft let alone a standard)
- http://www.w3.org/TR/2014/NOTE-ldp-acr-20140916/ (a note only)
- https://github.com/duraspace/pcdm/wiki#webacl
- Authorization Delegates
- http://www.w3.org/ns/auth/acl
- Hydra implementation of WebAC
    - https://github.com/projecthydra/hydra-head/blob/master/hydra-access-controls/app/models/hydra/access_controls/permission.rb
    - https://github.com/projecthydra/hydra-head/blob/master/hydra-access-controls/app/models/hydra/access_controls/access_control_list.rb
    - https://github.com/projecthydra/hydra-head/wiki/Access-Controls-with-Hydra