Authentication and Authorization

Overview

The Fedora Authentication (AuthN) and Authorization (AuthZ) framework is designed to be flexible and extensible, to allow any organization to configure access to suit its needs.

The following sections explain the Fedora AuthN/Z framework, and provide instructions for configuring some out-of-the-box access controls.

For clarity's sake, a distinction is made between Authentication and Authorization:

- **Authentication** answers the question "who is the person, and how do I verify that they are who they say they are?" Fedora relies on the web servlet container to answer this question.
- **Authorization** answers the question, "does this person have permission to do what they want to do?". Fedora provides two different ways to answer this question:
  - **Bypass authorization:** Anyone who has authenticated through the web application container (Tomcat, Jetty, WebSphere, etc.) has permission to do everything – in effect all, authenticated users are superusers.
  - **WebAC authorization:** Authenticated users' access to resources is mediated by WebAC Access Control Lists stored in the repository.

Servlet Container Authentication Configuration

Fedora relies on its servlet container to provide authentication. User credentials are configured in your web application container, usually in a properties file or XML file. This document describes how to set up Fedora and either Tomcat or Jetty to enable HTTP Basic Authentication, using simple user files. Consult your web application server documentation for other ways to configure and manage users. Fedora can handle any user principal passed to it by the servlet container, as provisioned by any of the container's supported authentication mechanisms.

Container Roles

Fedora uses two container roles to determine its authorization behavior. The superuser role is **fedoraAdmin**. Users with this role are not subject to any further authorization checks, and thus can perform any operations on the repository. This is comparable to the **fedoraAdmin** superuser role in Fedora 3, used for Fedora 3 API-M operations. The regular user role is **fedoraUser**. Users with this role *are* subject to authorization checks by the Web Access Control system. The exact permissions any regular user has are determined per request by looking at the effective ACL of the requested resource, the requesting user's security principals, and the nature of the request (HTTP method, content-type, etc.).

Configure Container Users and Roles

Jetty

- Create a **$JETTY_BASE/etc/jetty-users.properties** file. This file contains entries in the format *username: password [, role, ...]*, where
  - *username* is the user's login id (the principal)
  - *password* is the user's password
  - *role* is the servlet role they are assigned upon login; jetty allows you to specify any number of roles (or no role at all).
- Sample **jetty-users.properties** file that contains three users, two of whom are regular users, and the third of whom (fedoraAdmin) is a Fedora superuser:

jetty-users.properties

```
testuser: password1,fedoraUser
adminuser: password2,fedoraUser
fedoraAdmin: fedoraAdmin,fedoraAdmin
```

- Configure your Jetty login realm.
  - **Standalone**: Modify your **$JETTY_BASE/webapp/fcrepo.xml** file to configure the login realm and include the **jetty-users.properties** file:

    jetty.xml login service

    ```
    <Configure class="org.eclipse.jetty.webapp.WebAppContext">

      <!-- Set this to the webapp root of your Fedora repository -->
      <Set name="contextPath">/</Set>
      <!-- Set this to the path of of fcrepo4 WAR file -->
      <Set name="war"><SystemProperty name="jetty.base" default="."/>/webapps/fcrepo.war</Set>

      <Get name="securityHandler">
        <Set name="loginService">
          <New class="org.eclipse.jetty.security.HashLoginService">
            <Set name="name">fcrepo</Set>
            <!-- Set this to the path to your jetty-users.properties file -->
            <Set name="config"><SystemProperty name="jetty.base" default="."/>/etc/jetty-users.
    properties</Set>
          </New>
        </Set>
      </Get>

    </Configure>
    ```

  - **Embedded in Maven:** The fcrepo-webapp Maven project includes jetty-maven-plugin. The property *jetty.users.file* sets the location of the **jetty-users.properties** file. Run the fcrepo-webapp server with the following system property:

    ```
    -Djetty.users.file=/path/to/jetty-users.properties
    ```

- See the Jetty Authentication documentation for more details.

Tomcat

- Create or edit your `$CATALINA_HOME/conf/tomcat-users.xml` file. It has entries of the form

  ```
  <user name="principal" password="password" roles="role1, role2, ..." />
  ```

  where:

  - *name* is the user's login id (the principal)
  - *password* is the user's password
  - *roles* are the servlet roles they are assigned upon login; tomcat allows you to specify any number of roles (or no role at all).

  Sample **tomcat-users.xml** file that contains three users, two of whom are regular users, and the third of whom (fedoraAdmin) is a Fedora superuser:

  tomcat-users.xml

  ```
  <tomcat-users>
    <role rolename="fedoraUser" />
    <role rolename="fedoraAdmin" />
    <user name="testuser" password="password1" roles="fedoraUser" />
    <user name="adminuser" password="password2" roles="fedoraUser" />
    <user name="fedoraAdmin" password="fedoraAdmin" roles="fedoraAdmin" />
  </tomcat-users>
  ```

- Configure your Tomcat login realm. Modify your file `$CATALINA_HOME/conf/server.xml` file to configure the login realm with the Fedora webapp context:

server.xml

```
<Context>
  ...
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase" />
  ...
</Context>
```

- See the Tomcat Realms documentation for more details.

Bypass Authorization

Running Fedora without authorization means that the REST API is available to any request coming from the container and lacks any finer-grained security. This is useful when Fedora is running behind another application that connects to Fedora and implements its own security checks. This configuration is also useful for temporary demonstrations and for running software tests that do not require security.

Disabling authorization in Fedora does not preclude the use of container authentication to secure Fedora. However, container roles are not used for any further authorization within Fedora. All requests are treated as superusers.

To disable authorization simple set the `fcrepo.auth.enabled` configuration property to `false`, using either a configuration file or `-D` argument.

WebAC Authorization

Web Access Control (WebAC or WAC) is Fedora's system for authorizing requests for resources in the repository.

Definitions

From the SOLID Web Access Control specification:

> *Web Access Control (WAC) is a decentralized cross-domain access control system. The main concepts should be familiar to developers, as they are similar to access control schemes used in many file systems. It's concerned with giving access to agents (users, groups and more) to perform various kinds of operations (read, write, append, etc) on resources. WAC has several key features:*
>
> 1. *The resources are identified by URLs, and can refer to any web documents or resources.*
> 2. *It is* declarative *-- access control policies live in regular web documents, which can be exported/backed easily, using the same mechanism as you would for backing up the rest of your data.*
> 3. *Users and groups are also identified by URLs (specifically, by WebIDs)*
> 4. *It is* cross-domain *-- all of its components, such as resources, agent WebIDs, and even the documents containing the access control policies, can potentially reside on separate domains. In other words, you can give access to a resource on one site to users and groups hosted on another site.*

WebAC enforces access control based on the Access Control List (ACL) RDF resource associated with the requested resource. In WebAC, an ACL consists of a set of Authorizations. Each Authorization is a single rule for access, such as "users alice and bob may write to resource foo", described with a set of RDF properties. Authorizations have the RDF type `http://www.w3.org/ns/auth/acl#Authorization`.

For the remainder of this document, the `http://www.w3.org/ns/auth/acl#` namespace will be abbreviated with the prefix `acl:`.

Access Control Lists (ACLs)

An ACL is an RDF document (RDFSource) that contains WebAC statements that authorize access to repository resources. Each resource may have their own ACL, or implicitly be subject to the ACL of a parent container. The location of the acl for a given resource may be discovered via a `Link` header with relation `rel=acl`.

```
$ curl -I http://localhost:8080/fcrepo/rest/myContainer

Date: Thu, 23 Aug 2018 14:46:46 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
ETag: W/"919bed096330d23b2e85c01d487758aa6bbf2dcb"
Last-Modified: Thu, 16 Aug 2018 18:49:54 GMT
Link: <http://www.w3.org/ns/ldp#Resource>;rel="type"
Link: <http://www.w3.org/ns/ldp#Container>;rel="type"
Link: <http://www.w3.org/ns/ldp#BasicContainer>;rel="type"
Link: <http://localhost:8080/fcrepo/rest/myContainer/fcr:acl>; rel="acl"
Preference-Applied: return=representation
Vary: Prefer

...
```

If a resource does not have an individual ACL (and therefore relies on an implicit ACL from a parent), this link header will still be present, but will return a 404. This is because the location of ACLs is solely determined by the server, much like the automatically-created LDP-RS descriptions for binary resources. The key difference is that Fedora does not create ACLs automatically, only their location.

Therefore, to discover whether a resource has an individual ACL, a client would need to:

1. Perform a `HEAD` or `GET` against the resource,
2. Find the link header
3. Do a `GET` or `HEAD` against the ACL location, and see if returns 200 or 404.

To create an ACL for a resource that does not already have one, a client needs to discover the ACL location (via `HEAD` or `GET`), then `PUT` to that location.

Authorizations

An ACL should contain one or more authorizations. Each authorization should have a hash URI resource as its subject, and an `rdf:type` of `http://www.w3.org/ns/auth/acl#Authorization`:

Authorization

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

<#auth1> a acl:Authorization .
```

The properties that may be used on an `acl:Authorization` are:

Property	Meaning
`acl:accessTo`	The URI of the protected resource.
`acl:accessToClass`	An RDF class of protected resources. (*While the WebAC specification does not support acl:accessToClass, servers are required to support it according to the Fedora specification)*
`acl:agent`	The user *(in the W3C WebAC ontology, the user is named with a URI, but Fedora's implementation supports both URI- and string-based usernames)*
`acl:agentClass`	A class of agents, rather than a specific agent. Usage according to the WebAC specification is limited to `foaf:Agent` (meaning "everybody"), and `acl:AuthenticatedAgent` (meaning "any authenticated agent").
`acl:agentGroup`	A group of users (defined as a `vcard:Group` resource listing its users with the `vcard:hasMember` property).
`acl:default`	Signifies that an authorization for a container may be inherited by children of that container, if they do not otherwise define their own ACLs.
`acl:mode`	The type of access (WebAC defines several modes: `acl:Read`, `acl:Write`, `acl:Append`, and `acl:Control`).

For a more detailed explanation of Authorizations and their properties, see WebAC Authorizations.

Agents

Agents are the users of Fedora. These identify the principals (in a security sense) that have made authenticated requests to the repository. In ACL Authorizations used by Fedora, these may be represented as strings or as URIs. The SOLID WebAC spec stipulates that agents are identified by URIs, and suggests (but does not have any normative language requiring) that these URIs are intended to be WebIDs. The Fedora specification does not comment on the topic of identifying agents. Nevertheless, for legacy purposes, the Fedora 5.x software allows strings or URIs to identify agents (e.g. `"bob"` or `<http://example.org/people/bob>`). When using URIs, there is no expectation by Fedora that these URIs be resolvable, or have a representation. *It is highly recommended that you use URIs.*

The mapping of a logged-on principal to a string or URI depends on the selection and configuration of a Principal Provider, which may provide the identity of users as strings or URIs depending on its implementation. Because agents are recommended to be represented as URIs, Fedora can be configured to automatically prefix any principals that are provided as strings with a baseURI. This is achieved by setting the system property `fcrepo.auth.webac.userAgent.baseUri`. For example:

agent prefix

```
fcrepo.auth.webac.userAgent.baseUri=http://example.org/agent/
```

Continuing with this example, if a user comes in as user "dra2", the user's identity will be converted to the URI http://example.org/agent/dra2 before applying ACLs.

Examples of Authorizations

1. The user userA can Read document foo

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

<#auth1> a acl:Authorization ;
    acl:accessTo </fcrepo/rest/foo> ;
    acl:mode acl:Read;
    acl:agent "userA" .
```

2. Users in NewsEditor group can Write to any resource of type ex:News

```
@prefix acl: <http://www.w3.org/ns/auth/acl#> .
@prefix ex: <http://example.org/ns#> .

<#auth2> a acl:Authorization ;
    acl:accessToClass ex:News ;
    acl:mode acl:Read, acl:Write;
    acl:agentClass </fcrepo/rest/agents/NewsEditors> .
```

/agents/NewsEditors

```
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .

<> a vcard:Group;
    vcard:hasMember "editor1", "editor2".
```

3. The user userB can Read document foo (This involves setting a system property for the servlet container, e.g. `-Dfcrepo.auth.webac.userAgent.baseUri=http://example.org/agents/`)

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

<#auth3> a acl:Authorization ;
    acl:accessTo </fcrepo/rest/foo> ;
    acl:mode acl:Read;
    acl:agent <http://example.org/agents/userB> .
```

Protecting Resources

Any resource in the repository may have its own ACL. The location of that (potential) ACL is given in a `Link` HTTP header with `rel="acl"`. If a resource itself does not specify its own ACL, its parent containers are inspected, and the first specified ACL found is used as the ACL for the requested resource. If no ACLs are found, a filesystem-based ACL will be checked, the default policy of which is to deny access to the requested resource.

The standard location for a resource's ACL is the `fcr:acl` child of that resource, but clients should not rely on this behavior and always "follow their nose" by checking the `Link` header.

How-To Guides

- Quick Start with WebAC
- How to Use WebAC Groups
- WebAC Example Scenarios

More Detailed Documentation

- SOLID WebAC Specification
- Determining the Effective Authorization Using WebAC
- W3C's WebAC Ontology