Campus Shibboleth Metadata Management Tool

Local Campus Metadata Management Tool

One of the items identified early in our discussions on what is needed to make Shibboleth easier to deploy and use effectively on campus was the notion of a campus local metadata management tool. Some schools have created local solutions while many others manage local metadata by hand. The purpose of this document is to sufficiently define the high-level requirements for a TIER campus-level local metadata management tool such that we can solicit contributions from members or request development quotes from vendors.

The focus of the TIER Campus Metadata Management tool is to provide automation for the common, general campus use case and will not delve into specialized needs that will not be experienced at most campuses or with most entity descriptions. The tool focuses on Shibboleth SPs and should be trivial for a campus to brand and operate.

User Level Functional Requirements

1. End Users
    a. Enable campus users to authenticate, using Shibboleth, and request that their SP be added to the campus metadata distribution.
        i. Simple web form to prompt the SP operator for standard/simplified information. Built into the default web form is text that explains the needed elements and where to get the information.
        ii. Error checking on the entered data.
    b. The ability for the person who created the entry and their designates to edit the entry (including the list of designates).
    c. The ability for the identity that created the entry and designates to delete the entry.
    d. Ability to check on the processing status of the request, including the ability for pure self-service - i.e., automatic addition to the metadata.
    e. Potentially the ability to submit an XML blob for an entity. This blob would not be parsed or validated – it would simply be appended to the metadata file after approval by the system operators. Campus must be able to disable this option.
2. System Operators
    a. The ability for campus system operators to edit the information stored about any entity.
    b. The ability for campus system operators to approve/reject requests.
    c. The ability for campus system operators to change the ownership (e.g., the 1.c data) for any entity.
    d. The ability to ingest the full XML for an entity as a blob. This functionality is not meant to imply that the blob be parsed or validated. The requirement is simply that the blob be aggregated into the campus metadata file on build. Note that this can be a solution for Item 2.e above. A web interface is not required for 2.d whereas it would be needed for 1.e.
3. Information Requested (see Table 1)
    a. Contacts in Metadata - definitions for contact data as per InCommon

Table 1 – Information Requested Summary

Index	Element Name	Element Description
1	entityID	Used for entity id and binding creation
2	Certificates	Copy/paste boxes for certificate PEM
3	Bindings	Include some commonly used bindings with check boxes that users can select/un-select if needed. Default “on” is POST. Warnings should be displayed for back-channel and SLO use as they are not supported by the TIER distribution. A table-driven approach in the application is preferred. x HTTP-POST • o PAOS • o HTTP-POST-SimpleSign • o HTTP-Artifact • o SOAP • o HTTP-Redirect • o browser-post

4	Mdui	Prefer that this section support multiply occuring values. 1 DisplayName 2 Description 3 InformationURL 4 PrivacyStatementURL 5 Logo
5	Technical Contacts Information	Name, email, eppn (for ownership)
6	Administrative Contacts Information	Name, email, eppn (for ownership)
7	Security Contact	Name, email, eppn (for ownership)
8	Requested Attributes	A list of attribute names, space separated
9	Software Implementation	x Shibboleth SP • o SimpleSAMLphp • o Other _____
10	Notes	Is a notes field really needed?

Technical Requirements

1. General
    a. The tool only needs to handle the use case of Shibboleth SP metadata.
    b. The tool will send email to the user making a change and the listed contacts whenever the data for an entity is modified.
    c. The tool should be trivially simple to containerize, install, brand, and operate.
    d. The tool will provide some mechanism to track of previous versions of entity metadata. This requirement can be met by either literally archiving old metadata files or logging every change made with the date, time, and login identifier of the user making the change.
    e. The tool will be configurable to publish metadata (a) on a regular cycle, (b) after testing by a sysadmin, or (c) in semi-real time after a new record is submitted.
    f. (nice to have) The tool will provide support for regular service provider review process where a user authenticates and signs off that the entity record is up to date and still needed.
    g. (nice to have) Ability to know that sets of entities are linked (e.g., production and test). *This requirement is not well specified and should be considered for a future version.*
    h. The application should be architected to not preclude the addition of a future simple REST API that enables the creation, modification, and deletion of entity records.
    i. (Nice to have) The application validates the generated metadata file with a tool such as Oxygen to verify schema/syntax.
2. Architecture
    a. While other deployment models are anticipated, the most common model will be inside a Docker container, effectively behind a campus Shibboleth SP. Deployment using a Shibboleth SP should be assumed, though not required.
    b. Authentication and authorization data will be acquired external to the application and made available to the application in its environment. The application will use the identity and authorization attributes from the server environment or headers, including but not limited to REMOTE_USER. Access control must be possible through arbitrary combinations of attributes and values.
        i. The application should be architected and delivered using pluggable Authentication and Authorization modules. The interface between these modules and the application should be as simple as practical to meet the application's needs.
        ii. Per-entity authorization data is maintained and consumed by these modules.
    c. User interface components are not hard coded into the application.
3. Application Behavior with the default, provided AuthN/AuthZ modules (i.e., the out-of-the box behavior of the application)
    a. Campus administrators will have the ability to use some combination of simple user identity (eppn), groups, entitlements, etc., depending on their infrastructure, to control access to entity records.
    b. The default, provided, AuthN/Z modules will:
        i. Obtain authnz data from the environment
        ii. Via a simple configuration mechanism that can be a web interface, a script to set database values, or a flat file, provide the campus administrator with the ability to configure access control for entity metadata based on a list of attributes.
            1. If any of these attributes has a value that matches the attribute data stored with the entity's metadata, access to the record is granted.
            2. For clarification, one way to implement this feature would be to store name/value pairs with the entity metadata record. If the user for the current transaction matches any of these stored name/value pairs, they are authorized to manage the record.

iii. Via a simple configuration mechanism that can be a web interface, a script to set database values, or a flat file., provide the campus administrator with the ability to specify a list of name/value pairs (e.g., EPPNs, groups, etc.) for administrative access.
iv. Users with administrative access have the ability to edit any/all entity records

c. Log messages should always include the environment's configured user value.

4. Data Management
    a. The application is free to store is data using any reasonable mechanism. For example, a standard SQL database could work well for this application. Simply using the filesystem to store blobs that can be parsed could also work well for this application.
    b. Whatever database/file-system mechanism is used, it needs to support simple backups. These backups do not need to be part of the application.