Shibboleth Metadata Management GUI

High Level Project Description

Background

One of the traditional impediments to smaller school adoption of the Shibboleth IdP has been its XML-based configuration management. Recent work by the Shibboleth team for TIER has expanded the ability to control the IdP configuration via entity attributes ("tags") associated with the SAML metadata of a relying party. These enhancements will become part of the Shibboleth distribution with the next release. The goal for the work has been to enable metadata-based control of 80% or more of the routine day-to-day configuration changes made by administrators.

The Shibboleth enhancements were designed to facilitate the implementation of a GUI to manage configuration while, to the greatest extent possible, minimize the interactions between the GUI and Shibboleth itself. The Shibboleth documentation on this feature in the form in which it will be made available in V3.4 is here, and this proposal assumes V3.4 features as the basis of the work.

The GUI's initial role will be to support a significant range of metadata configuration options, essentially supplying either a generated metadata configuration using the existing Shibboleth configuration syntax for metadata, or to supply alternative MetadataResolver implementations whose underlying content is managed via the GUI. This includes support for cases in which SAML metadata already exists as well as the ability to produce the metadata for specific services via the GUI.

Examples of administrative tasks the GUI will support are described in a companion document (TBD).

Non-Requirements

For the initial phase of work, it is not a requirement to support the configuration of any other aspects of the IdP through the GUI. This is simply out of recognition that additional features will begin to encroach on much more complicated parts of the system's configuration and require further coordination with the Shibboleth Project to define appropriate boundaries or insulation techniques to allow for both GUI- and hand-managed settings.

It may turn out that other areas of the system are badly in need of improvement and it may also turn out to be simpler than expected to facilitate the use of a GUI, but it is believed that at least 80% or more of day to day tasks may be offloaded by management (and generation) of metadata alone and starting there is the best way to prove the viability of any GUI as a part of the software ecosystem, given the need for long term support and maintenance of such a project.

It is also not an initial requirement to support metadata or registry formats other than SAML metadata, and it is likely that Shibboleth support for other protocols will frequently include profiles of SAML metadata in any case, due to it being flexible enough to encompass any protocol and because the existing APIs in the software tend to assume it.

Relationship to Existing Software

There are existing software products specifically dedicated to producing and managing SAML metadata, but do not include actual support for managing the Shibboleth metadata configuration itself. It may be advantageous to evaluate extending an existing product or at least borrow UI or code from one.

It may also be appropriate to discuss development or adoption of pieces of code by the Shibboleth Project in the form of libraries and APIs to reduce the amount of code maintained as part of the GUI.

High-Level Goals

A “mission statement” for the GUI is to avoid the need for deployers to routinely modify the metadata or relying party configuration of the IdP software, and (provided that some up front planning and work is done) also reduce the frequency of manipulation of the rest of the configuration, through the use of metadata properties to drive SAML NameID behavior and attribute release policy.

The overarching goal is to support the addition of third-party metadata sources (e.g., federations such as InCommon) with appropriate security and policy controls, and the provisioning of and configuration of IdP behavior for Service Providers that do not support the use of federations as a means of acquiring their metadata.

Detailed Requirements

Functional Requirements

Required application functionality includes:

1. The ability to create, configure, and maintain SAML metadata sources based on all of the non-deprecated Shibboleth MetadataResolver types, with the exception of the “ChainingMetadataProvider” and “ResourcedBackedMetadataProvider” types. All non-deprecated required and optional settings should be supported, subject to exceptions proposed by the implementer and discussed with the Shibboleth Project.
2. The ability to generate locally maintained SAML SP metadata via direct input of specific fields (i.e., endpoints, keys), to include at minimum:
    a. EntityID
    b. Certificates(s) containing keys for signing, encryption, or both
    c. Endpoints for profiles supported by the IdP, including Browser SSO, ECP, and Single Logout via any of the standard SAML bindings supported by the IdP
    d. Enumeration of required NameID Format(s)
    e. The AuthnRequestsSigned and WantAssertionsSigned boolean attributes
    f. MDUI extensions (Logo, DisplayName, etc.)
    g. Organization and ContactPerson elements
3. The ability to add, configure, and maintain all of the non-deprecated MetadataFilter types. All non-deprecated required and optional settings should be supported, subject to exceptions proposed by the implementer and discussed with the Shibboleth Project.

4. A built-in set of options governing the behavior of the IdP that are controllable through the use of various MetadataFilter types, principally the “EntityAttributes” filter and the property conventions proposed by the Shibboleth Project (referred to subsequently as “built-in properties”).
    a. Provide a mechanism to attach descriptive information to built-in properties, including hyperlinks to documentation.
5. An extensible set of named properties and values (referred to subsequently as “extended properties”). The built-in properties should be viewed as special cases of extended properties.
    a. Provide a mechanism to view, create new, and edit existing extended properties and their possible values. The property value types supported must include:
        i. String
        ii. Boolean
        iii. Integer
        iv. Long
        v. Double
        vi. Duration
        vii. List of any of the primitive types
        viii. Set of any of the primitive types
        ix. Spring Bean ID
    b. Provide a mechanism to import additional extended property definitions.
    c. Provide a mechanism to attach descriptive information to extended properties, including hyperlinks to documentation.
6. Mechanisms to create, view, edit, delete, and duplicate associations between built-in and extended properties and the SP(s) to which they should be associated, along with an optional comment.
    a. Associations may be accomplished through direct insertion of <EntityAttributes> extensions into SP metadata, or through the attachment of an EntityAttributes metadata filter (which allows for more concise representation of associations of properties to many SPs at once). Associations should be possible on the basis of at least:
        i. Default/global behavior
        ii. Specific SP(s) based on name or regular expressions
        iii. SPs whose metadata includes existing <EntityAttributes> extensions (e.g. entity categories such as the Research and Scholarship designation).
        iv. (optional) SPs for which a Java scriptlet evaluates successfully (this is a documented feature of the metadata filter added to V3.4).
7. Generation of a Shibboleth metadata configuration for use by the IdP. Generation of a file is an obvious approach, but an alternative approach of implementing a new MetadataResolver plugin may be discussed.
    a. The configuration must be confirmed as valid by instantiating an instance of the IdP’s MetadataResolver service facade and verifying a successful outcome.
    b. Review by an administrator must be possible before application of a configuration, both visually and by means of some form of query functionality through the MetadataResolver API to verify aspects of the metadata.
8. Archive each version of the generated configuration, and provide for labeling of versions and explicit or policy-based removal of older versions (e.g., versions older than a date, or beyond a certain number).
9. An audit record of changes and additions.
10. Data and transactional integrity in the face of concurrent access by multiple administrators (see also Technology Requirements).

Technical/Platform Requirements

The GUI is expected to be a web application that can be deployed into many environments, including stand-alone and into the TIER VM and Docker container distributions.

Other requirements:

1. The primary programming language must be Java.
2. Selection of platform and UI framework is subject to approval by TIER.
3. Assuming a servlet-based design as a foundation, support for Jetty and Tomcat versions supported by the Shibboleth Project at the time of authoring are required, and the use of proprietary container features must be avoided.
4. The application must rely on a transaction-safe relational database, preferably via a standard abstraction layer (i.e., JDBC) to facilitate support for multiple databases. Support for PostgreSQL, and MariaDB (with an appropriate back-end for transactional integrity), are required.
5. A cross-platform installation and upgrade process, with upgrades retaining existing configuration.

The technical infrastructure should be designed to minimize deployment and operational requirements. The application will be used occasionally as opposed to continually, so trade-offs to optimize ease of deployment and operation over performance are appropriate. Scalability in the face of the amount of data is a requirement, but concurrent load can be assumed to be low.

Security Requirements

The application should be expected to be operated for internal use of a small number of dedicated staff, and as such does not require an extensive security model.

Appropriate modern best practices in protecting against common threats such as XSS should be followed. Under no circumstances should any Javascript usage rely on code acquired from any server other than the server hosting the application.

Specific requirements:

1. Authentication via REMOTE_USER, servlet attribute, or request header (based on configuration). For auditing purposes, an authenticated identity is a requirement.
2. An authorization model based on matching REMOTE_USER, servlet attribute, or request header names and values. A simple OR semantic allowing for any of a set of rules to grant access is sufficient (more complex requirements can be left to means such as use of Apache).

It is optional to support finer-grained authorization, but a logical separation may be between basic and advanced modes of operation (see User Interface Requirements).

User Interface Requirements

The target audience for this application is primarily deployers who are seeking a simpler experience, so ease of use and comprehension are the driving factors. As such, many of the functional requirements are likely more appropriate to an “advanced” mode or view that is hidden from the entry-level administrator in

favor of a more basic mode that supports the most rudimentary approach, namely direct provisioning of individual SPs with their own settings. Features involving extended properties and arbitrary assignment of settings to groups of SPs via indirect criteria do not need to be accessible in this mode.

Other requirements:

1. Compliance with Section 508 of the Americans with Disabilities Act.
2. A template-based UI that allows for customization of header/footer content, logo, and general style settings without code customizations. It is desirable to support these changes at runtime (i.e., without repackaging/restart of the application) to simplify debugging of changes.
3. Internationalization (i18n) without changing application source code, with language selection driven by standard HTTP language negotiation. Actual translations of the UI are not required. In the event that the Spring Framework is used, standard Spring i18n features, message properties, etc. must be used.
4. Appropriate pagination and search capabilities in the face of large numbers of data elements (e.g., SPs and metadata sources, extended properties).

Miscellaneous Requirements

Other requirements:

1. Deployment and user documentation should be provided, accessible online and in a context-sensitive manner during use of the application.
2. Release of the source code under the Apache 2.0 license, or a compatible license approved by the TIER Component Architects committee.
3. Use of dependent libraries available only under the Apache 2.0 license, or a compatible license approved by the TIER Component Architects committee.
4. Use of the Internet2 Enterprise GitHub as the official Source Code Repository.

Project and Development Support

TIER will provide:

1. Current TIER distributions of the IdP for integration testing and validation.
2. A set of default extended properties designed around the default IdP configuration to be supplied by TIER with respect to supported SAML NameID and Attribute settings. For example, TIER may define a set of “standard” attributes to be supported by the IdP given for example a “typical” LDAP directory, and thus define a set of extended properties corresponding to these attributes which could map to simple lists or checkboxes of attributes to release to SPs.
3. Access to, and communication of, any intended TIER modifications to the default Shibboleth distribution, to ensure alignment.
4. Assistance with testing.
    a. SZ: This needs to be qualified/specified/narrowed

The Shibboleth Project will provide:

1. Example metadata configurations upon request for any of the required or optional features of the tool and agenda time as required on our development calls to address questions.
2. Review of metadata configurations produced by the tool.
3. Due consideration of any recommended changes or additions to the default IdP configuration in light of the development of the tool.