SHRINE 4.1.0 Appendix A.8 - More Details: Using Authorization with SSO

Core Authorization Configuration:

If you want to use ***authorization*****,** you must first add the following configuration to ***shrine.conf*****,** after the existing shrine block**:**

```
shrine {

...

}
...
shrine.config.authorizer.requireAuthorization = "true"
shrine.webclient.ssoLogoutUrl = "https://<your hostname>/shrine-api/authorizer/logout"
shrine.config.authorizer.shibLogoutUrl = "https://<your hostname>/Shibboleth.sso/Logout?return=<return URL
provided by your idP>"
// shrine.webclient.unauthorizedMessage = "Enter your message"
```

Unauthorized Message:

The default unauthorized message is as follows and currently baked into the code: ***"You currently do not have access to SHRINE. Please contact your institution's SHRINE administrator for more information."***

*(**Optional**)* The unauthorized message can be tailored to your needs in ***shrine.conf*** by uncommenting and updating the message:

```
// shrine.webclient.unauthorizedMessage = "Enter your message"
```

(Required) Authorization Logic Configuration:

Authorization has 2 phases:

Phase 1: Collecting "attributes" about the user. **Note: the user is identified by the REMOTE_USER / userId header passed by the SP – See section A. 3**

Phase 2: Making an authorization decision based on the attributes collected in Phase 1

The authorization system works with any number of individually configured (**Phase 1**) **attribute providers,** each of which can generate attributes. Further, a single (**Phase 2**) **authorization provider**, must also be configured. The authorization provider will determine, based on the collected attributes, whether the user is authorized or not.

> *NOTE: After the configuration items indicated above in the* **shrine.conf** *config file, we also need to add a configuration block called '* **shrine.config.authorizer'**.
>
> **The following configuration pattern is used to integrate attribute providers with the authorization provider. The system currently comes with 3 available AttributeProviders and 3 available Authorization Providers. However, the system can be configured with any number of AttributeProviders but only one of the AuthorizationProvider should be configured and be used.**

```
// Configuration for Phase 1 (attribute providers) and Phase 2 (one authorizer)
//
shrine.config.authorizer : {

  attributeProviders :  // this example uses three attribute providers -- there must be a non-empty list
  [
        {...} // configuration for an available attribute provider
    {...} // configuration for an available attribute provider
    {...} // configuration for an available attribute provider
   ],
  authorizer : {        // exactly one authorization provider must be configured
    ...   // configuration for an available authorization provider
  }
}
```

The attribute providers will assemble attributes in a data structure with the following form: Each "attribute type" corresponds to an AttributeProvider class. Each AttributeProvider class generates a list of "attributes", and each of the attributes has a list of values. The authorizer class will use this data to decide whether to authorize the user or not.

```
// Structure of attribute-collection generated in Phase 1
* {
   *   attribute type 1 -> {
   *     attribute 1 -> [value 1, value 2, ...],
   *     attribute 2 -> [value 1, value 2, ...],
   *     ...
   *   },
   *   attribute type 2 -> {
   *     attribute 1 -> [value 1, value 2, ...],
   *     attribute 2 -> [value 1, value 2, ...],
   *     ...
   *   },
   *   ...
   * }
```

Attribute Providers Configuration:

WhiteBlackListAttrProvider:

The WhiteBlackListAttrProvider queries a database's table of whitelisted and blacklisted users. Its typical configuration follows. It finds the user by looking for the REMOTE_USER / userId passed by the SP.

```
    {
      class = net.shrine.authz.providerService.attributes.WhiteBlackListAttrProvider
      name = wb-list,
      // DB config here should correspond to tomcat's Resource in its context.xml, see below
      database: {
        dataSourceFrom = "JNDI"
        jndiDataSourceName = "java:comp/env/jdbc/blackWhiteTableDB"
        timeout = "30 seconds"
        createTablesOnStart = false
      }
    }
```

WhiteBlackList Context.xml Configuration

Note that the table and column names are not configurable. The db table must be named "bw_user" and the columns "ssoId", "whitelisted", and "blacklisted". As configured above, the database name is "blackWhiteTableDB"; but it could be configured to another name. Also, the context.conf file in the tomcat configuration must contain the following:

```
<Resource name="jdbc/blackWhiteTableDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="128" maxIdle="32" maxWaitMillis="10000"
              username="shrine" password="<shrine-pw>" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/blackWhiteTableDB?serverTimezone=UTC"
              testOnBorrow="true" validationQuery="SELECT 1"/>
```

The WhiteBlackListAttrProvider generates attributes of this shape:

```
    wb-list: -> {
              isBlack -> (true/false},
              isWhite -> (true/false}
                }
```

EndpointAttrProvider

An EndpointAttrProvider fetches data from a remote URL and extracts attributes from that data by using Regexes. In the example which follows it extracts 2 attributes, person_id and faculty_type:

```
    {
      class = net.shrine.authz.providerService.attributes.EndpointAttrProvider
      name = profiles_faculty_type_and_id
      url = ".....{userId}....."
      userIdPlaceHolder="{userId}" // the REMOTE_USER / userId will get substituted into the Url for this
placeholder
      attributeRegexes : [
        {
          name = "person-id"
          regex = "PersonID=\"([0-9]+)\""
        }
        {
          name = "faculty_type"
          regex = "<Affiliation Primary=\"true\">.*?FacultyTypeSort=\"(.)\""
        }
      ]
    }
```

The attributes generated by an EndpointAttrProvider as configured above will have this shape:

```
profiles_faculty_type_and_id -> {
   person-id: [...]
   faculty_type: [...]
}
```

One can re-use EndpointAttrProvider in the same **shrine.config.authorizer** configuration. For example, the same attribute provider class could be configured as follows

```
    {
      class = net.shrine.authz.providerService.attributes.EndpointAttrProvider
      name = endpoint_everything
      url = ".....{userId}....."
      userIdPlaceHolder="{userId}" // the REMOTE_USER / userId will get substituted into the Url for this
placeholder
      attributeRegexes : [
        {
          name = "everything"
          regex = "(.+)"
        }
      ]
    }
```

The attributes generated by an EndpointAttrProvider as configured above will have this shape, where "everything" will contain the entire payload from the call to the 3rd party end-point

```
endpoint_everything -> {
  everything: [...]
}
```

RequestHeadersAttrProvider

The RequestHeadersAttrProvider extracts values from HTTP request headers:

```
    {
      class = net.shrine.authz.providerService.attributes.RequestHeadersAttrProvider
      name = headers,
      headerNames :
        [
           AJP_userId
           AJP_email
           AJP_firstName
           AJP_lastName
        ]
      }
```

The attributes generated by RequestHeadersAttrProvider as configured above will have this shape:

```
headers -> {
                AJP_userId: [...]
        AJP_email: [...]
        AJP_firstName: [...]
        AJP_lastName: [...]
}
```

Authorization Providers Configuration:

HmsAuthorizer

The authorization provider, for example, HmsAuthorizer, makes use of the attributes generated by the attribute providers. Per the requirements for HMS, HmsAuthorizer checks a 'Profiles' endpoint.

```
  // You are authorizied if and only if:
  //         You are not black-listed
  // --and-- you are either white-listed or your faculty type is from 0 to 4 inclusive

  authorizer : {
    name : net.shrine.authz.providerService.authorize.HmsAuthorizer
  }
```

RegexAuthorizer

A more flexible authorization provider could be the RegexAuthorizer. It concatenates all the received attributes and values, and then applies any number of Regexes to it. Authorization is granted if all regexes find a match. A "!" before a Regex means that there should not be a match.

```
// Given the below regexTerms, you are authorizied if and only if:
//         You are not black-listed
// --and-- you are either white-listed or your faculty type is from 0 to 4 inclusive
// --and-- the string 'fp77' does not appear anywhere in your attributes

authorizer : {
    name : net.shrine.authz.providerService.examples.RegexAuthorizer
    regexTerms :
        [
           "wb-list.isBlack.false"
           "(wb-list.isWhite.true)|(faculty_type_and_id.faculty_type.[0-4])"
           "!(fp77)"
        ]

  }
```

BWAuthorizer

With BWAuthorizer, authorization is granted if the user is white-listed and NOT black-listed.

```
// You are authorizied if and only if you are white-listed but NOT black-listed

authorizer : {
    name : net.shrine.authz.providerService.authorize.BWAuthorizer
  }
```

Next Step:

SHRINE 4.1.0 Appendix A.9 - Starting and Stopping the Software