Shibboleth Identity Provider (IdP)

Sebastian Rieger
sebastian.rieger@gwdg.de

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany

CLARIN AAI Hands On Workshop, 25.02.2009, Oxford eResearch Center
Integrating your local identities in federations

- setting up your own IdP allows your users to access services (SPs) in federations and to cooperate with other institutions

- the IdP reflects the “home organization” of your users in the federation, it ensures proper authentication (username, password) and authorization, credentials and sensitive data are stored in the institute (not at the SPs)

- on successful authentication the IdP issues a digitally signed assertion (token, ticket), that authenticates the user

- a reference to the assertion is stored as a Cookie (HTTP session) in the user’s web browser

- by redirecting the client to the IdP (using HTTP redirect or POST), while accessing another SP, assertion (Cookie) is used to authenticate the user without further logins → Single Sign-On is achieved as long as assertion is valid
Architecture of a Shib IdP

- Shibboleth IdP is a Java Web Application, Download [http://shibboleth.internet2.edu](http://shibboleth.internet2.edu) JDK 1.6 preferred, installation script (install.sh)
- Java Security Provider and Tomcat configuration
- platform independent Linux, Windows …

Documentation:
- [http://www.switch.ch/aai/support/howto/](http://www.switch.ch/aai/support/howto/)

Basic Components:

- Web Server (Apache e.g. 2.2)
- Application Server (Tomcat e.g. 5.5)

- IdP is deployed in idp.war (current version 2.1.2)
  - **AuthN**: SSOHandler used for authentication: HTTP, form-based or existing session / assertion (Cookie), certificates in development, using AuthN DB or LDAP, custom implementations possible (JAAS)
  - **AuthZ**: Attribute Authority allowing AuthZ using attributes, using Attribute DB, LDAP, …

Source: [http://www.switch.ch/aai/demo/](http://www.switch.ch/aai/demo/)
Configuring Apache and Tomcat

- setting up a virtual host in Apache, passing access to local Tomcat
- using certificate and keyfile of the IdP

/etc/apache2/conf.d/idp.conf:

```xml
<VirtualHost 192.168.0.1:443>
  ServerName my-new-idp.example.com:443
  SSLEngine on
  SSLCertificateFile /opt/shibboleth-idp/credentials/idp.crt
  SSLCertificateKeyFile /opt/shibboleth-idp/credentials/idp.key

  <Proxy ajp://localhost:8009/idp/*> Allow from all </Proxy>
  ProxyPass /idp/ ajp://localhost:8009/idp/
</VirtualHost>

Listen 8443

<VirtualHost 192.168.0.1:8443>
  ServerName my-new-idp.example.com:8443
  SSLEngine on
  SSLCertificateFile /opt/shibboleth-idp/credentials/idp.crt
  SSLCertificateKeyFile /opt/shibboleth-idp/credentials/idp.key

  <Location /idp>Allow from all SSLOptions +StdEnvVars +ExportCertData
    SSLVerifyClient optional_no_ca SSLVerifyDepth 10 </Location>
  <Proxy ajp://localhost:8009/idp/*> Allow from all </Proxy>
  ProxyPass /idp/ ajp://localhost:8009/idp/
</VirtualHost>
```
Configuring Apache and Tomcat

- setting up the Tomcat to deploy the idp.war
- remember to reserve enough Java heap (using –Xmx, -Xms) (e.g. /etc/default/tomcat5.5)

server.xml

```xml
<Server port="8005" shutdown="SHUTDOWN">
  <!-- Define the Tomcat Stand-Alone Service -->
  <Service name="Catalina">
    <!-- Define an AJP 1.3 Connector on port 8009 -->
    <Connector port="8009" address="127.0.0.1"
      request.tomcatAuthentication="false" enableLookups="false"
      redirectPort="8443" protocol="AJP/1.3" />
  </Service>
</Server>
```

/etc/tomcat5.5/Catalina/localhost/idp.xml

```xml
<Context docBase="/opt/shibboleth-idp/war/idp.war" privileged="true"
  antiResourceLocking="false" antiJARLocking="false" unpackWAR="false" />
```
**a Shib 2.0 session initialization in detail**

Demo showing each HTTP request / response

Source: [http://www.switch.ch/aai/demo/](http://www.switch.ch/aai/demo/)
Connecting the IdP to Identity Management

- IdP can use a directory service (LDAP, AD, ...), database, custom connectors
- Attribute Resolver and AuthN engine can use separate configurations
- Two integration types for AuthN: external (web server, or tomcat realm) and internal (form-based auth using Shibboleth)

**Example: “internal” Shib-based AuthN via LDAP** (login.config)

```plaintext
ShibUserPassAuth { edu.vt.middleware.ldap.jaas.LdapLoginModule sufficient
    host="ldap.gwdg.de"
    port="636"
    base="dc=gwdg,dc=de"
    ssl="true"
    userField="uid"
    subtreeSearch="true"; }

// alternate Kerberos config
ShibUserPassAuth { com.sun.security.auth.module.Krb5LoginModule sufficient; }
```

- Failover config possible by inserting more “sufficient” configurations (fall-through)
Connecting the IdP to Identity Management

- Configuring AuthN handler in handler.xml

```xml
<!-- Shib "internal" Username/password login handler -->
<LoginHandler xsi:type="UsernamePassword" jaasConfigurationLocation="file:///opt/shib-idp/conf/login.config">
  <AuthenticationMethod>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
  </AuthenticationMethod>
</LoginHandler>

<!-- alternate handler for external AuthN e.g. Tomcat - using HTTP REMOTE_USER header -->
<LoginHandler xsi:type="RemoteUser">
  <AuthenticationMethod>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified
  </AuthenticationMethod>
</LoginHandler>
```

- failover for IdP possible by using Load Balancing (Layer 3), Cost: no SSO after failover
  - Session-aware Load Balancing (Layer 7) supported since IdP 2.1 using terracotta (“network attached memory” to provide persistent clustered heap, see tc-config.xml)

Other services (besides SAML-based profiles) defined in the handler.xml:

- Status (display the status of the IdP) e.g. https://shib-idp.mpg.de/idp/profile/Status
- Metadata (retrieve metadata from IdP) e.g. https://shib-idp.mpg.de/idp/profile/Metadata/SAML
- Login Handler „PreviousSession“ to allow SSO
Joining federations with your IdP

- Federations / trusts can be assigned in relying-party.xml
  - multiple federations can be assigned
  - definition of the IdP entity and certificate & private key used for digital signatures

```xml
<DefaultRelyingParty provider="https://shib-idp.mpg.de/idp"
  defaultSigningCredentialRef="IdPCredential">
  Publish service URLs (profiles) of the IdP
    - SSO (AuthN)
    - AttributeQuery (handling requests for attributes to do AuthZ)
    - ArtifactResolution (requesting an assertion using an Artifact e.g. POST profile)
    - all handlers are offered for SAML 2.0 and SAML 1.1 to ensure interoperability with Shibboleth 1.1 (by the way… end of life 6/30/2010!)
  automatic download and caching of the metadata of the assigned federations
```
Joining federations with your IdP cont.

- Configuring integration of metadata in relying-party.xml (e.g. trusted SPs)

```xml
// get metadata.xml from URL, cache it locally, check signature
// and ignore parties not being SPs
<MetadataProvider id="URLMD" xsi:type="FileBackedHTTPMetadataProvider"
 xmlns="urn:mace:shibboleth:2.0:metadata"
 metadataURL="http://www.aai.dfn.de/fileadmin/metadata/DFN-AAI-Test-metadata.xml"
 backingFile="/opt/shibboleth-idp/metadata/DFN-AAI-Test-metadata-backingFile.xml">
  <MetadataFilter xsi:type="ChainingFilter" xmlns="urn:mace:shibboleth:2.0:metadata">
    <MetadataFilter xsi:type="SignatureValidation"
                 xmlns="urn:mace:shibboleth:2.0:metadata"
                 trustEngineRef="shibboleth.MetadataTrustEngine" requireSignedMetadata="true" />
    <MetadataFilter xsi:type="EntityRoleWhiteList"
                   xmlns="urn:mace:shibboleth:2.0:metadata">
      <RetainedRole>samlmd:SPSSODescriptor</RetainedRole>
    </MetadataFilter>
  </MetadataFilter>
</MetadataProvider>

// security trust engine to check metadata signature as configured in the filter
<security:TrustEngine id="shibboleth.MetadataTrustEngine"
 xsi:type="security:StaticExplicitKeySignature">
 <security:Credential id="MyFederation1Credentials" xsi:type="security:X509Filesystem">
   <security:Certificate>/opt/shibboleth-idp/credentials/dfn-aai.pem</security:Certificate>
 </security:Credential>
</security:TrustEngine>
```
Multiple Federations

- joining multiple federations can be accomplished in several ways
  - register the IdP in multiple federations and configure them in relying-party.xml
  - usage of bridging protocols (e.g. eduGAIN) building a confederation

- for MPG-AAI we developed an IdP Proxy that is placed in multiple federations and allows different authentication mechanisms

- Implementation as custom JAAS module (external Tomcat AuthN), custom data connector

- users select the single IdP Proxy in the Discovery Service and use their local identities (and IdPs), to log in using their mail address, IdP is derived from the domain / realm

- attributes going to external federations can be filtered

- only one IdP for the entire MPG in the external DS
the installation script of the IdP creates a keypair, this can be used for further requests

```
root@idp:/opt/shibboleth-idp/credentials# openssl req -new -key idp.key -out idp.csr
```

resulting CSR can be sent to your NREN CA, issued certificate is copied to idp.crt

path to files is configured in relying-party.xml and the virtual host, containing your IdP e.g.:

```
Listen 8443

<VirtualHost 192.168.0.1:8443>
    ServerName idp.example.com:8443
    SSLEngine on
    SSLCertificateFile /opt/shibboleth-idp/credentials/idp.crt
    SSLCertificateKeyFile /opt/shibboleth-idp/credentials/idp.key
    <Location /idp>
        Allow from all
    </Location>
    <Proxy ajp://localhost:8009/idp/*> Allow from all </Proxy>
    ProxyPass /idp/ ajp://localhost:8009/idp/
</VirtualHost>
```

Java keystore IdP.jks can be deleted, or you can import cert and key using keytool
Interaction with SPs...

- IdP delivers assertions and attributes to SPs, Shibboleth allows a finegrained filtering of the outgoing attributes

- Attributes are based on standards: inetOrgPerson, eduPerson, dfnEduPerson
  - e.g. eduPersonAffiliation, eduPersonTargetedId, eduPersonEntitlement

Attribute management at the IdP is split into two instances: Resolver and Filter

- Resolver gets attributes from directory services, databases etc. (attribute-resolver.xml)
  - Attributes are converted and mapped e.g. to URN (e.g. urn:geant:dfn.de:mpg: ...)
  - Combining multiple attribute (sources), and complex formatting are possible

- Filter defines Attribute Release Policy (ARP) for consuming SPs / federations (attribute-filter.xml)
  - Filtering can be configured individually for every SP / federation the IdP knows
  - Complex filter scenarios are possible (filtering specific values, defining filter rules)
Attribute Resolver

- first data connectors are defined as sources for the attributes in attribute-resolver.xml...

<!-- Example Static Connector -->

```xml
<resolver:DataConnector id="staticAttributes" xsi:type="Static"
    xmlns="urn:mace:shibboleth:2.0:resolver:dc">
    <Attribute id="eduPersonAffiliation"><Value>member</Value></Attribute>
    <Attribute id="eduPersonEntitlement">
        <Value>urn:mace:dir:entitlement:common-lib-terms</Value>
    </Attribute>
</resolver:DataConnector>

<!-- Example LDAP Connector -->

```xml
<resolver:DataConnector id="myLDAP"
    xsi:type="LDAPDirectory", xmlns="urn:mace:shibboleth:2.0:resolver:dc"
    ldapURL="ldaps://ldap.gwdg.de" baseDN="O=GOESTERN"
    principal="cn=roadmin,o=GWDG", principalCredential="secret">
    <FilterTemplate>
        <![CDATA[
            (uid=$requestContext.principalName)
        ]]>
    </FilterTemplate>
</resolver:DataConnector>

- failover LDAP servers possible by separating different URLs with a blank
Attribute Resolver cont.

<!-- Example Relational Database Connector -->
<resolver:DataConnector id="myDB" xsi:type="RelationalDatabase"
xmlns="urn:mace:shibboleth:2.0:resolver:dc">
  <ApplicationManagedConnection
      jdbcDriver="oracle.jdbc.driver.OracleDriver"
      jdbcURL="jdbc:oracle:thin:@db.example.org:1521:SomeDB"
      jdbcUserName="myid"
      jdbcPassword="mypassword" />
  <QueryTemplate>
    <![CDATA[
      SELECT * FROM student WHERE gzbtpid = $requestContext.principalName
    ]]>}
  </QueryTemplate>
  <Column columnName="gzbtpid" attributeID="uid" />
  <Column columnName="fqlft" attributeID="gpa" type="Float" />
</resolver:DataConnector>

<!-- StoredID (persistentID) Connector e.g. to generate eduPersonTargetedId -->
<resolver:DataConnector id="myStoredId" xsi:type="StoredId"
xmlns="urn:mace:shibboleth:2.0:resolver:dc" generatedAttributeID="persistentID"
sourceAttributeID="uniqueID" salt="some/secret">
  <resolver:Dependency ref="uniqueID" />
  <ApplicationManagedConnection
      jdbcDriver="com.mysql.jdbc.Driver"
      jdbcURL="jdbc:mysql://localhost:3306/shibboleth?autoReconnect=true"
      jdbcUserName="user" jdbcPassword="secret" />
</resolver:DataConnector>
Attribute Resolver cont.

- the attributes from the data connectors have to be mapped to SAML attributes

```xml
<resolver:AttributeDefinition id="eduPersonAffiliation" xsi:type="Simple"
xmlns="urn:mace:shibboleth:2.0:resolver:ad"
sourceAttributeID="eduPersonAffiliation">
  <resolver:Dependency ref="staticAttributes" />
  <resolver:Dependency ref="myLDAP" />
  <resolver:AttributeEncoder xsi:type="SAML1String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder",
name="urn:mace:dir:attribute-def:eduPersonAffiliation" />
  <resolver:AttributeEncoder xsi:type="SAML2String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder",
name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1" friendlyName="eduPersonAffiliation" />
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="eduPersonEntitlementOvid" xsi:type="Template"
xmlns="urn:mace:shibboleth:2.0:resolver:ad">
  <resolver:Dependency ref="IdPProxyDataConnector" />
  <resolver:AttributeEncoder xsi:type="SAML1String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder",
name="urn:mace:dir:attribute-def:eduPersonEntitlement" />
  <resolver:AttributeEncoder xsi:type="SAML2String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder",
name="urn:oid:1.3.6.1.4.1.5923.1.1.1.7" friendlyName="eduPersonEntitlement" />
  <Template><![CDATA[urn:wkhmr:ovid.institutional.login:${uid}]></Template>
</resolver:AttributeDefinition>

- other complex attribute types are possible, several dependencies can be defined (e.g. for failover)
Attribute Filter

- filters defined in attribute-filter.xml - deny attributes and values for federations / SPs ...

```xml
<AttributeFilterPolicy>
  <PolicyRequirementRule xsi:type="basic:OR">
    <basic:Rule xsi:type="saml:AttributeRequesterInEntityGroup"
      groupID="urn:mace:incommon" />
    <basic:Rule xsi:type="saml:AttributeRequesterInEntityGroup"
      groupID="https://www.aai.dfn.de/DFN-AAI-Test" />
  </PolicyRequirementRule>
  <AttributeRule attributeID="eduPersonScopedAffiliation">
    <PermitValueRule xsi:type="basic:ANY" />
  </AttributeRule>
  <AttributeRule attributeID="eduPersonAffiliation">
    <PermitValueRule xsi:type="basic:OR">
      <basic:Rule xsi:type="basic:AttributeValueString" value="faculty"/>
      <basic:Rule xsi:type="basic:AttributeValueString" value="student"/>
    </PermitValueRule>
  </AttributeRule>
</AttributeFilterPolicy>

<AttributeFilterPolicy>
  <PolicyRequirementRule xsi:type="basic:AttributeRequesterString",
    value="https://shibboleth.ovid.com/entity" />
  <AttributeRule attributeID="eduPersonEntitlementOvid">
    <PermitValueRule xsi:type="basic:ANY" />
  </AttributeRule>
</AttributeFilterPolicy>
```
Ensuring privacy...

- You can use aacli.sh to check which attributes are delivered by the IdP

- Filtering attributes and their values is not enough to ensure privacy, personal data belonging to the user is maybe not be allowed to leak from the institute...

- but SPs need at least an attribute to differentiate between users
  - solution is to deliver anonymized id for the user → eduPersonTargetedId
  - persistent id being a hash over „username + idp secret + sp name“, irreversible, users get different ids for different SPs (service publishers can not track the users)

- for other attributes containing personal data, users can get the possibility to explicitly approve attributes being sent: SWITCHaai uApprove

- Users will need to accept terms of use and confirm the sending of attributes to every SP they use, reconfirmation is needed if policy, or the set of delivered attributes changes

→ Demo: using Ovid service in the MPG-AAI
Customizing idp login etc.

- login page can be customized in the IdP sources – see src/main/webapp/login.jsp
- also login-error.jsp (e.g. for failed authentication) can be customized
- error.jsp, error404.jsp (failures during usage of the profiles and handlers)
- changes can be deployed by using install.sh again (builds the WAR and deploys it)
Logging

- Shibboleth IdP logging is configured in logging.xml
- accesses are logged to /opt/shibboleth-idp-2.0.0/logs/idp-access.log
- and /opt/shibboleth-idp-2.0.0/logs/idp-audit.log
- debugging information can be taken from /opt/shibboleth-idp-2.0.0/logs/idp-process.log

```xml
<!-- Logs IdP, but not OpenSAML, messages -->
<logger name="edu.internet2.middleware.shibboleth">
    <level value="DEBUG" />
</logger>

<!-- Logs OpenSAML, but not IdP, messages -->
<logger name="org.opensaml">
    <level value="DEBUG" />
</logger>

<!-- Logs LDAP related messages -->
<logger name="edu.vt.middleware.ldap">
    <level value="DEBUG" />
</logger>
```

- consider privacy of logged information! → Demo
Outlook: IdP in Shibboleth 2.x

- Shib IdP 2.2
  - Single Logout functionality (SAML-conform SLO, partly already implemented)
  - ECP profile of SAML 2.0 (Enhanced Client or Proxy / Delegation e.g. SOAP, …)
  - X.509 and other authentication mechanisms (Kerberos, SPNEGO, ADFS etc. – currently available using plug-ins)
  - native support for privacy enhancements like uApprove
- integration of user-centric Identity Management (CardSpace, OpenID)?
- maybe support for desktop applications? e.g. using ECP?

Thanks for your attention! any questions?