Using OIOSAML for Java

Installation and configuration

Version: 3.2.1

Table of Contents

[Table of Contents 2](#_Toc93916122)

[1 OIOSAML.java 4](#_Toc93916123)

[1.1 Overview 4](#_Toc93916124)

[1.2 Version 3.0.0 4](#_Toc93916125)

[2 Mandatory Configuration 4](#_Toc93916126)

[2.1 DispatcherServlet 4](#_Toc93916127)

[2.2 AuthenticatedFilter 4](#_Toc93916128)

[3 How to configure OIOSAML 5](#_Toc93916129)

[3.1 Configure through web.xml 5](#_Toc93916130)

[4 Configuration Parameters 7](#_Toc93916131)

[4.1 DispatcherServlet 7](#_Toc93916132)

[4.2 AuthenticatedFilter 14](#_Toc93916133)

[4.3 DispatcherServlet Configuration from File 15](#_Toc93916134)

[4.4 Multiple AuthenticatedFilters and Step-Up 15](#_Toc93916135)

[5 Additional Configuration 17](#_Toc93916136)

[6 Maven Dependency 17](#_Toc93916137)

[7 Demo Application 18](#_Toc93916138)

[7.1 oiosaml.properties 18](#_Toc93916139)

[7.2 web.xml 18](#_Toc93916140)

[7.3 private.jsp 18](#_Toc93916141)

[7.4 Running the demo application 18](#_Toc93916142)

[8 Logging 19](#_Toc93916143)

[9 Audit Logging 20](#_Toc93916144)

[9.1 Implementing custom AuditLogger 20](#_Toc93916145)

[9.2 Configuring Audit logging 20](#_Toc93916146)

[10 OIOSAML Session handling 21](#_Toc93916147)

[10.1 In memory session handling 21](#_Toc93916148)

[10.2 Database session handling 21](#_Toc93916149)

[10.2.1 Sample JNDI configuration 22](#_Toc93916150)

[10.2.2 Sample JDBC configuration 22](#_Toc93916151)

[10.3 Configuration of session cleanup 22](#_Toc93916152)

[11 Test Identity Provider 22](#_Toc93916153)

[11.1 Configuration 22](#_Toc93916154)

[11.2 Compilation 22](#_Toc93916155)

[11.3 Execution 23](#_Toc93916156)

# 1 OIOSAML.java

This document covers how to use and configure the OIOSAML.java 3.x framework, which is intended for use with NemLog-in3, but it also be used as a generic SAML framework for Java.

The framework supports the OIO SAML 3.0 profile and can generate AuthnRequests and perform validation on SAML Assertions according to this profile.

## 1.1 Overview

The framework consists of two primary classes, which must be configured to use the framework. These are

* dk.gov.oio.saml.servlet.DispatcherServlet
* dk.gov.oio.saml.filter.AuthenticatedFilter

The first is a Servlet implementation, that must be exposed on /saml/\*, so it can manage all SAML related requests.

The second is a Filter implementation, that should be placed in front of any web-resources that requires the user to be authenticated. It will ensure that the user goes through the SAML login process before they can access the web-resource that the Filter is placed in front of.

The OIOSAML.Java framework works with Java 8+.

## 1.2 Version 3.0.0

Version 3.0.0 was released before NemLog-in3 was ready for use, so implementation and testing where performed against NemLog-in2. Therefore, it is not certain that version 3.0.0 will function optimally against NemLog-in3.

It is recommended to upgrade to version 3.0.1 when such a version becomes available.

# 2 Mandatory Configuration

Both the DispatcherServlet and the AuthenticatedFilter requires some minimum configuration before they can be used. The full configuration of both classes is described in further chapters in this document, but for quick setup, the following is the only required settings

## 2.1 DispatcherServlet

The DispatcherServlet must be configured with the following information before it can be used

* KeyStore configuration, used for signing AuthnRequests and decrypting SAML responses
* SAML EntityID for the application
* BaseURL to be used for SAML metadata generation
* IdP metadata reference (URL or File)

With these settings in place, the DispatcherServlet will function. More configuration is possible, as documented in the following chapters.

## 2.2 AuthenticatedFilter

The AuthenticatedFilter have no mandatory configuration settings. If not configured, it will use the default settings as documented in the following chapters.

# 3 How to configure OIOSAML

All configuration is done through the Servlet and Filter classes, and the configuration is supplied through the FilterConfig and ServletConfig classes, supplied when creating a Servlet or Filter in Java.

How this is done, depends on the specific application framework used together with OIOSAML.Java, but the most common way is to configure Servlets and Filters through a web.xml deployment file, as shown below

# 3.1 Configure through web.xml

If the Java application contains a web.xml deployment file, this can be used to configure any Servlets and Filters in the application, including OIOSAML.java.

The DispatcherServlet can be configured in the following way (see the chapter with configuration parameters below for which keys and values can be used in the init-param sections)

<servlet>

<servlet-name>DispatcherServlet</servlet-name>

<servlet-class>

dk.gov.oio.saml.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>CONFIG\_KEY\_1</param-name>

<param-value>CONFIG\_VALUE\_2</param-value>

</init-param>

<init-param>

<param-name>CONFIG\_KEY\_1</param-name>

<param-value>CONFIG\_VALUE\_2</param-value>

</init-param>

<init-param>

… etc …

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

And then deployed with the following section

<servlet-mapping>

<servlet-name>DispatcherServlet</servlet-name>

<url-pattern>/saml/\*</url-pattern>

</servlet-mapping>

Similarly, the AuthenticatedFilter can be configured and deployed like this

<filter>

<filter-name> AuthenticatedFilter </filter-name>

<filter-class>

dk.gov.oio.saml.filter.AuthenticatedFilter

</filter-class>

<init-param>

<param-name>CONFIG \_KEY\_1</param-name>

<param-value>CONFIG\_VALUE\_1</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>AuthenticatedFilter</filter-name>

<url-pattern>/protected/\*</url-pattern>

</filter-mapping>

Configuration of the SessionDestroyListener also need to be added, so that sessions destroyed by the server are also removed from the OIOSAML Session handler.

The configuration is as follows:

<listener>

<listener-class>

dk.gov.oio.saml.session.SessionDestroyListener

</listener-class>

</listener>

# 4 Configuration Parameters

This chapter describes all the configuration parameters, and their default values

## 4.1 DispatcherServlet

|  |  |  |  |
| --- | --- | --- | --- |
| **Setting** | **Mandatory** | **Default Value** | **Description** |
| oiosaml.servlet.entityid | Yes |  | Must be filled out with the EntityID which identifies the application as a Service Provider  e.g. [http://saml.serviceprovider.com](http://saml.serviceprovider.com/) |
| oiosaml.servlet.baseurl | Yes |  | Must be filled out with the URL on which the application is accessible in a web-browser. The value is used to generate SAML metadata which must contain login/logout URL endpoints.  e.g. [https://serviceprovider.com](https://serviceprovider.com/)  the value above will result in metadata generated URLs like the one below  <https://serviceprovider.com/saml/assertionConsumer> |
| oiosaml.servlet.keystore.location | Yes |  | This must be filled out with the name of the PKCS#12 keystore file, located on classpath of the application |
| oiosaml.servlet.keystore.password | Yes |  | This must be filled out with the password to the PKCS#12 keystore given above. |
| oiosaml.servlet.keystore.alias | Yes |  | This must be filled out with the alias of the keyentry in the PKCS#12 keystore given above |
| oiosaml.servlet.idp.entityid | Yes |  | This must be filled out with the EntityID of the SAML Identity Provider that is used for login |
| oiosaml.servlet.idp.metadata.file | Partially |  | This can be filled out with a FILE reference to the SAML Identity Provider metadata.  The file must be located on the classpath of the application.  Note that either a FILE or URL reference is required. |
| oiosaml.servlet.idp.metadata.url | Partially |  | This can be filled out with a URL reference to the SAML Identity Provider metadata.  Note that either a FILE or URL reference is required. |
| oiosaml.servlet.configurationfile | No |  | This can be filled out with a FILE reference to a configuration file.  If supplied, the DispatcherServlet will read its configuration from that file instead of the ServletConfiguration section.  More details below. |
| oiosaml.servlet.profile.validation.enabled | No | true | By default, the OIOSAML.Java framework performs OIO SAML 3.0 profile validation.  If this is not needed, turn of this setting by setting the value to “false”. |
| oiosaml.servlet.profile.validation.assurancelevel.allowed | No | false | The NemLog-in IdP cannot for all authentication provide a NSIS LoA. Therefore, the service provider can decide to accept the AssuranceLevel which the NemLog-in IdP will provide instead.  If this is not acceptable turn off this setting either by omitting it or setting the value to “false”. |
| oiosaml.servlet.profile.validation.assurancelevel.minimum | No | 3 | If the AssuranceLevel is acceptable a minimum value can be specified using this setting.  Any integer is accepted for this setting, however the NemLog-in IdP will never provide an integer larger than 3. |
| oiosaml.servlet.metadata.nameid.format | No | urn:oasis:names:tc:SAML:2.0:nameid-format:persistent | By default, the OIOSAML.Java framework generates SAML metadata that contains a requested NameID format with this value.  To change the value, enter another SAML NameID Format. |
| oiosaml.servlet.metadata.contact.email | No |  | By default, the OIOSAML.Java framework does not generate a Contact section in the SAML metadata.  Supply an email address here if a Contact section is needed. |
| oiosaml.servlet.idp.metadata.refresh.min | No | 1 | By default, the OIOSAML.Java framework will attempt to refresh the Identity Provider metadata sometime between 1 and 12 hours.  Change this value to change the lower bound of the interval with which the framework refreshes the SAML metadata from the IdP. |
| oiosaml.servlet.idp.metadata.refresh.max | No | 12 | By default, the OIOSAML.Java framework will attempt to refresh the Identity Provider metadata sometime between 1 and 12 hours.  Change this value to change the upper bound of the interval with which the framework refreshes the SAML metadata from the IdP. |
| oiosaml.servlet.secondary.keystore.location | No |  | This value is configured like the primary keystore above but allows for adding a second certificate to the SAML metadata, allowing for changing the certificate with low to no downtime.  Note that this requires the SAML Identity Provider to support multiple certificates. |
| oiosaml.servlet.secondary.keystore.password | No |  | This value is configured like the primary keystore above but allows for adding a second certificate to the SAML metadata, allowing for changing the certificate with low to no downtime.  Note that this requires the SAML Identity Provider to support multiple certificates. |
| oiosaml.servlet.secondary.keystore.alias | No |  | This value is configured like the primary keystore above but allows for adding a second certificate to the SAML metadata, allowing for changing the certificate with low to no downtime.  Note that this requires the SAML Identity Provider to support multiple certificates. |
| oiosaml.servlet.signature.algorithm | No | http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 | By default, the RSA SHA256 algorithm is used for signing. Change this value if another algorithm is needed. |
| oiosaml.servlet.secondary.page.error | No |  | The OIOSAML.Java framework has a build-in error page, which is shown in case of SAML related errors.  Set this value to have the OIOSAML.Java framework redirect the user to another webpage in case of errors.  See more details with regards to getting error information in the section below. |
| oiosaml.servlet.secondary.page.logout | No |  | The OIOSAML.Java framework redirects the user to the context root of the web-application after a successful logout.  Set this value to a webpage that OIOSAML should redirect the user to instead of the context root. |
| oiosaml.servlet.secondary.page.login | No |  | When the login process completed, the OIOSAML.Java framework will attempt to redirect the user to the web-resource that they tried to access before the login process started.  If this fails, the OIOSAML.Java framework will try to redirect the user to the page specified by the value. If no value is specified the context root of the application is used. |
| oiosaml.servlet.trust.selfsigned.certs | No | false | By default, the OIOSAML.Java framework performs certificate validation when accessing HTTPS protected resources like SAML metadata.  Set this value to “true” to disable certificate validation. |
| oiosaml.servlet.revocation.crl.check.enabled | No | true | By default, the OIOSAML.Java framework performs revocation checking using OCSP and CRL checking.  Set this value to “false” to disable CRL revocation checking. |
| oiosaml.servlet.revocation.ocsp.check.enabled | No | true | By default, the OIOSAML.Java framework performs revocation checking using OCSP and CRL checking.  Set this value to “false” to disable OCSP revocation checking. |
| oiosaml.servlet.routing.path.prefix | No | saml | Routing configuration, servlet path prefix for the OIO dispatch servlet.  Change this to change where the IOISAML 3 endpoint is mounted in the application context (oiosaml.servlet.baseurl).  Ex.  Default “/saml/logout”, from “/{oiosaml.servlet.routing.path.prefix}/{oiosaml.servlet.routing.path.suffix.logout}” to hit the logout action |
| oiosaml.servlet.routing.path.suffix.error | No | error | Routing configuration, servlet error path |
| oiosaml.servlet.routing.path.suffix.metadata | No | metadata | Routing configuration, servlet metadata path |
| oiosaml.servlet.routing.path.suffix.logout | No | logout | Routing configuration, servlet logout path |
| oiosaml.servlet.routing.path.suffix.logoutResponse | No | logoutResponse | Routing configuration, servlet logoutResponse path |
| oiosaml.servlet.routing.path.suffix.assertion | No | assertionConsumer | Routing configuration, servlet assertionConsumer path |
| oiosaml.servlet.audit.logger.classname | No | dk.gov.oio.saml.audit.Slf4JAuditLogger | Class reference for the AuditLogger used in the project. See documentation under “Audit logging” |
| oiosaml.servlet.audit.logger.attribute.ip | No | request:remoteAddr | Used in Audit logging to lookup and log the users IP address.  See documentation under “Audit logging” |
| oiosaml.servlet.audit.logger.attribute.port | No | request:remotePort | Used in Audit logging to lookup and log the users PORT.  See documentation under “Audit logging” |
| oiosaml.servlet.audit.logger.attribute.userid | No | request:remoteUser | Used in Audit logging to lookup and log the users UserId in the SP application.  See documentation under “Audit logging” |
| oiosaml.servlet.audit.logger.attribute.sessionId | No | request:sessionId | Used in Audit logging to lookup and log the users SessionId in the SP application.  See documentation under “Audit logging” |
| oiosaml.servlet.session.handler.factory | No | dk.gov.oio.saml.session.inmemory.InMemorySessionHandlerFactory | Class reference for the SessionHandlerFactory used when handling sessions in the OIOSAML project.  Following session handler factories already exists in the OIOSAML project:   * dk.gov.oio.saml.session.database.JdniSessionHandlerFactory * dk.gov.oio.saml.session.database.JdbcSessionHandlerFactory * dk.gov.oio.saml.session.inmemory.InMemorySessionHandlerFactory |
| oiosaml.servlet.session.handler.jdni.name | No |  | Configuration parameter for the JNDI session handler factory: dk.gov.oio.saml.session.database.JdniSessionHandlerFactory  JNDI name for the database, value example: jdbc:h2:mem:mydb |
| oiosaml.servlet.session.handler.jdbc.url | No |  | Configuration parameter for the JDBC session handler factory: dk.gov.oio.saml.session.database.JdbcSessionHandlerFactory  JDBC Connection URL for the database, value example: jdbc:mysql://@localhost:3306/oiosaml |
| oiosaml.servlet.session.handler.jdbc.username | No |  | Configuration parameter for the JDBC session handler factory: dk.gov.oio.saml.session.database.JdbcSessionHandlerFactory  JDBC Connection logon username, example value: username |
| oiosaml.servlet.session.handler.jdbc.password | No |  | Configuration parameter for the JDBC session handler factory: dk.gov.oio.saml.session.database.JdbcSessionHandlerFactory  JDBC Connection logon password, example value: password |
| oiosaml.servlet.session.handler.jdbc.driver.classname | No |  | Configuration parameter for the JDBC session handler factory: dk.gov.oio.saml.session.database.JdbcSessionHandlerFactory  JDBC java driver class name for the connection, example value for mysql: com.mysql.cj.jdbc.Driver  Remember to add dependencies in POM. |
| oiosaml.servlet.session.handler.inmemory.max.tracked.assertionids | No | 10000 | Configuration parameter for the in memory session handler factory: dk.gov.oio.saml.session.inmemory.InMemorySessionHandlerFactory  The maximum number of tracked Assertion IDs used to ensure that we are not replaying requests. Old elements will be removed first. |

## 4.2 AuthenticatedFilter

|  |  |  |  |
| --- | --- | --- | --- |
| **Setting** | **Mandatory** | **Default Value** | **Description** |
| oiosaml.filter.ispassive.enabled | No | false | If the filter should send AuthnRequests with isPassive=true, set this value to “true” |
| oiosaml.filter.forceauthn.enabled | No | false | If the filter should send AuthnRequests with forceAuthn=true, set this value to “true” |
| oiosaml.filter.nsis.required | No | NONE | If a minimum NSIS level is required, set the value to one of   * LOW * SUBSTANTIAL * HIGH   And the filter will generate an AuthnRequest that contains the required value and will validate that the issued Assertion contains at least this value.  When set to NONE, an issued Assertion with AssuranceLevel is implicitly accepted. |
| oiosaml.filter.attribute.profile | No |  | If a specific profile is required in the assertion from the IdP (Person or Professional), set this value to one of   * <https://data.gov.dk/eid/Person> * <https://data.gov.dk/eid/Professional>   And the filter will generate an AuthnRequest with the requested profile and perform validation on the issued Assertion. |

## 4.3 DispatcherServlet Configuration from File

If the “oiosaml.servlet.configurationfile” setting is supplied to the DispatcherServlet, it will read its configuration from the supplied file instead. This file should be an ordinary property file, supplying properties as key/value pairs like the example below

oiosaml.servlet.idp.entityid=https://saml.test-nemlog-in.dk/

oiosaml.servlet.idp.metadata.url=https://test-nemlog-in.dk/Testportal/Test-nemlog-in-2.xml

## 4.4 Multiple AuthenticatedFilters and Step-Up

It is possible to configure multiple AuthenticatedFilters, each with its own configuration. One common use-case for this, is requiring different NSIS levels for different resources in the application.

If both a LOW and a SUBSTANTIAL filter is configured, and the user tries to access a resource protected by the LOW filter, then the AuthnRequest will inform the Identity Provider that the user only needs to perform a NSIS LOW login. If the user then later tries to access a resource protected by the SUBSTANTIAL filter, it will perform a step-up, and ask the Identity Provider to increase the NSIS level by performing additional authentication processes with the user.

## 4.5 AppSwitch Extension

When using the AppSwitch extension for a iOS or Android app the following configuration parameters should be defined.

|  |  |  |  |
| --- | --- | --- | --- |
| **Setting** | **Mandatory** | **Default Value** | **Description** |
| oiosaml.appswitch.returnurl.android | No | empty | Should contain the dynamic link for the android application. |
| oiosaml.appswitch.returnurl.ios | No | empty | Should contain the universal link for the iOS application. |

Triggering the AppSwitch extension is done by adding the “appSwitchPlatform=<PLATFORM>” parameter to the URL when starting an authentication flow. Where <PLATFORM> is either Android or iOS.

# 5 Additional Configuration

As SAML requires cross-domain configuration, and modern browsers by default prevents sending cookies on cross-domain requests, SAML login can under certain circumstances fail, because no session can be found when parsing the response from the SAML Identity Provider.

One solution is to ensure that the SameSite attribute is set to None when issuing sessions. If this can be done in a controlled fashion from the application itself, that is fine.

If not, the OIOSAML framework supplies a Filter that can be used for setting the SameSite attribute to True on all issued cookies.

**Note!**

If the SameSite attribute is already set to some value, the Filter will NOT modify the attribute. It will only set the SameSite attribute on cookies that do not already have that attribute.

The SameSite filter must be configured to run before any other filters that may create sessions. When using web.xml as a deployment mechanism, this is done by placing the SameSite filter before the other filters (e.g. above the AuthenticatedFilter in web.xml)

There is no configuration of the Filter, just deployment as shown in the example below

<filter>

<filter-name>SameSiteFilter</filter-name>

<filter-class>

dk.gov.oio.saml.filter.SameSiteFilter

</filter-class>

</filter>

<filter-mapping>

<filter-name>SameSiteFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

# 6 Maven Dependency

OIOSAML is released as a publicly available Maven dependency and can be included as shown below. Make sure to change the version number to the latest version.

<dependency>

<groupId>dk.digst</groupId>

<artifactId>oiosaml3.java</artifactId>

<version>3.0.1</version>

</dependency>

# 7 Demo Application

A demo application is supplied with the OIOSAML.java source code. The demo application shows how to use the OIOSAML framework and contains the following structure

├── config

│   ├── keystore.pfx

│   └── ssl-demo.pfx

├── pom.xml

├── src

│   └── main

│   ├── resources

│   │   ├── log4j2.properties

│   │   ├── oiosaml.properties

│   │   └── Test-nemlog-in-2.xml

│   └── webapp

│   ├── error.jsp

│   ├── index.jsp

│   ├── low

│   │   └── private.jsp

│   ├── nonsis

│   │   └── private.jsp

│   ├── substantial

│   │   └── private.jsp

│   └── WEB-INF

│   └── web.xml

The config folder contains the keystores used by the application.

The src folder contains everything else, with the following files being relevant.

## 7.1 oiosaml.properties

The demo application uses an external file for configuring the DispatcherServlet, and the configuration can be found here.

## 7.2 web.xml

The demo application shows how to configure the DispatcherServlet and AuthenticatedFilter (with step-up) in the web.xml file

## 7.3 private.jsp

Finally, there are three folders, each containing identical JSP files. These files are protected by the AuthenticatedFilter by various NSIS level requirements and will show the content of the issued SAML assertion.

## 7.4 Running the demo application

Compiling and running the demo application is performed with Maven like this

$ mvn clean install tomcat7:run-war

Note that Tomcat 7 performs some validation on class files during startup, which has some issues with JAXB. This results in warnings, that can safely be ignored. Tomcat 8 does not have these issues.

Once the application is running, it can be accessed here

<https://localhost:8443/oiosaml3-demo.java/>

It should show a page that looks like this

Text

Description automatically generated

The various links will either download the SAML metadata generated by the demo application or attempt to access one of the protected resources.

# 8 Logging

OIOSAML is using SLF4J as its logging framework, as do OPENSAML, but OPENSAML has dependencies that require JCL and Log4j.

To implement a logging framework in OIOSAML we need to bridge JCL and Log4j to SLF4J and choose an output logging framework.

Bridging JCL and Log4j to SLF4J is done by including these dependencies in the demo project:

<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.32</version>  
</dependency>  
  
<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>jcl-over-slf4j</artifactId>  
 <version>1.7.32</version>  
</dependency>  
  
<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>log4j-over-slf4j</artifactId>  
 <version>1.7.32</version>  
</dependency>

Customizing your logging solution is then done by selecting an implementation, if you need to customize logging to write log output to your own custom implementation of a logging framework, the simplest solution is to fork slf4j-simple (<https://github.com/qos-ch/slf4j/tree/master/slf4j-simple>) and write your own implementation of SimpleLogger (look at innerHandleNormalizedLoggingCall()).

To write logging using Log4j-2 like in the demo project just add this dependency:

<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-slf4j-impl</artifactId>  
 <version>2.14.1</version>  
</dependency>

Most logging frameworks has an SLF4J implementation or bridge.

Implementing logging in your project comes down to adding dependencies to forward logging from OIOSAML and its dependencies to your preferred logging framework.

# 9 Audit Logging

The audit logging must be persisted in 6 months and contain information relevant to use of the functionality exposed in the OIOSAML service.

The audit log is channeled through the AuditLogger interface which could be overwritten in configuration by providing you own implementation via the configuration property “oiosaml.servlet.audit.logger.classname”.

If you use the default SLF4J implementation it is important to configure logging so that the audit log output is persisted in 6 months.

The SLF4JAuditLogger implementation adds a “AUDIT” tag to log messages, that could be used to store audit logging separate from regular application logging.

## 9.1 Implementing custom AuditLogger

To replace the default SLF4J audit logger you can implement your own AuditLogger by providing an implementation of “dk.gov.oio.saml.audit.AuditLogger” in the configuration like this:

oiosaml.servlet.audit.logger.classname=dk.gov.oio.saml.audit.Slf4JAuditLogger

Take a look at “dk.gov.oio.saml.audit.SLF4JAuditLogger” for inspiration. To support a JPA based implementation just remember that we need a default constructor and that injection will not work out of the box.

## 9.2 Configuring Audit logging

You can customize content in the audit logging to your application for a few attributes: IP, Port, UserId and SessionId.

This is done by providing values for these configuration parameters:

oiosaml.servlet.audit.logger.attribute.ip=request:remoteAddr

oiosaml.servlet.audit.logger.attribute.port=request:remotePort

oiosaml.servlet.audit.logger.attribute.userid=request:remoteUser

oiosaml.servlet.audit.logger.attribute.sessionId=request:sessionId

The configuration values conform to the following EBNF syntax:

<value> ::= <protocol>:<attribute>

<protocol> ::= <query> | <header> | <cookie> | <session> | <request>

<attribute> ::= Name of an attribute accessible from the selected protocol.

<query> ::= Access to GET and Form POST query parameters/attributes.

<header> ::= Access to request Header names, as parameters/attributes.

<cookie> ::= Access to request Cookie names, as parameters/attributes.

<session> ::= Access to session values i.e. to access SessionId for logging.

<request> ::= remoteHost | remoteAddr | remotePort | remoteUser

(From the Servlet request)

To use the JSESSIONID cookie as SessionId in the Audit log add the following to your OIOSAML configuration:

oiosaml.servlet.audit.logger.attribute.sessionId=cookie:JSESSIONID

# 10 OIOSAML Session handling

Session handling in OIOSAML is configured in “oiosaml.properties” by providing a factory for creating the session handler and its configuration.

The class reference to the session handler factory is provided by setting the configuration property:

* oiosaml.servlet.session.handler.factory

The following session handler factories are provided in the OIOSAML project:

* InMemorySessionHandlerFactory
* JdbcSessionHandlerFactory
* JdniSessionHandlerFactory

You can provide your own implementation based on the SessionHandlerFactory interface:

public interface SessionHandlerFactory {

void configure(Configuration config);

SessionHandler getHandler();

void close();

}

Take a look at the provided implementations for inspiration. You will need your own implementation if performance is an issue, i.e. to ensure that connections is provided by a connection pool and if you like to optimize the solution with caching.

## 10.1 In memory session handling

This is the default session handler.

The InMemory session handler is a simple implementation that stores all values on the HTTP session, only configuration property is:

* oiosaml.servlet.session.handler.inmemory.max.tracked.assertionids

That limits the size of the list containing requested Assertion IDs, which is there to prevent replay.

## 10.2 Database session handling

OIOSAML contain two database session handler factories for initializing session handling on a database using JNDI or JDBC.

Initializing the database to use the OIOSAML database session handler is done by executing the DDL script included in the /misc folder (database\_session\_handler.sql).

The script will create a database and tables for the database session handler.

The sample configuration below can be used on a default installation of mysql, if you start by executing the “database\_session\_handler.sql” script on the database (set passwords and create application users as needed).

### 10.2.1 Sample JNDI configuration

oiosaml.servlet.session.handler.factory=dk.gov.oio.saml.session.database.JdniSessionHandlerFactory

oiosaml.servlet.session.handler.jdni.name=jdbc:h2:mem:oiosaml

### 10.2.2 Sample JDBC configuration

oiosaml.servlet.session.handler.factory=dk.gov.oio.saml.session.database.JdbcSessionHandlerFactory

oiosaml.servlet.session.handler.jdbc.url=jdbc:mysql://@localhost:3306/oiosaml

oiosaml.servlet.session.handler.jdbc.username=root

oiosaml.servlet.session.handler.jdbc.password=

oiosaml.servlet.session.handler.jdbc.driver.classname=com.mysql.cj.jdbc.Driver

## 10.3 Configuration of session cleanup

OIOSAML have two cleanup jobs, one to ensure that inactive sessions are removed and one to ensure that HTTP sessions that has been destroyed also is removed from the OIOSAML session handler.

Both the session handler and the session cleaner service is initialized in OIOSAML3Service, but the SessionDestroyListener is set up in the demo project as part of the web.xml.

The configuration is as follows:

<listener>

<listener-class>

dk.gov.oio.saml.session.SessionDestroyListener

</listener-class>

</listener>

If you are unable to use this SessionDestroyListener as is, you will need to provide something similar, to remove sessions that has been invalidated outside OIOSAML.

# 11 Test Identity Provider

A test Identity Provider is supplied with the OIOSAML.Java source code. It is primarily used for manual testing when developing the OIOSAML.Java framework and should not be considered a production ready Identity Provider.

The Demo application can be used with the test Identity Provider by modifying the oiosaml.properties file in the demo application to point to the test Identity Provider. Commented out configuration should be available for easy access.

## 11.1 Configuration

In the config folder found in the IdP module, there is an application.properties file, containing the various settings for the IdP.

The important section is the “Test Cases”, which lists all the usernames and passwords which can be used with the test IdP. Each username has a set of attributes assigned to them, which can be modified here. They are used to test various OIOSAML profile scenarios.

## 11.2 Compilation

The test IdP requires Java 11 and is compiled using maven using the following Maven command.

$ mvn clean install

## 11.3 Execution

The test IdP is build using the Spring Framework, and can be started using the Spring Maven plugin like this

$ mvn spring-boot:run

When running, the IdP is available on port 7080 (the port can be changed in application.properties).