Spring Security SAML Extension

Reference Documentation

Vladimír Schäfer

Spring Security SAML Extension: Reference Documentation

by Vladimír Schäfer

3.0-RC2

Table of Contents

Part I. Getting Started

This chapter provides essential information needed to enable your application to act as a service provider and interact with identity providers using SAML 2.0 protocol. Later in this guide you can find information about detailed configuration options and additional use-cases enabled by this component.

1. Introduction

1.1 What this manual covers

This manual describes Spring Security SAML SP component, it's uses, installation, configuration, design and tested environments.

1.2 When to use Spring Security SAML SP

Component enables both new and existing applications to act as a Service Provider in federations based on SAML 2.0 protocol. The key use-case beind the project is enabling web single sign-on. All products supporting SAML 2.0 in Identity Provider mode such as ADFS 2.0, OpenAM (OpenSSO), RM5 IdM or Ping Federate can be used to connect with Spring Security SAML SP.

Spring Security SAML SP can be either embedded inside application and work along other authentication or single sign-on mechanisms or it can be deployed separately and convey authentication information to applications using a custom mechanism.

Open-source alternatives to this component are e.g. native SAML service providers integrating with IIS or Apache from Shibboleth, where SAML processing is done on the web server and not on the application level. OpenAM Fedlet is another mean to enable SAML SP capabilities in an application.

1.3 Features and supported profiles

Current implementation should be conformant to SAML SP Lite and SAML eGovernment profile. The following profiles, bindings and features are supported as part of the product:

- Web single sign-on profile
- Web single sign-on holder-of-key profile
- IDP and SP initialized single sign-on
- Single logout profile
- Enhanced client/proxy profile
- Identity provider discovery profile and IDP selection
- Metadata interoperability and PKIX trust management
- Automatic service provider metadata generation
- Metadata loading from files, URLs, file-backed URLs
- Processing and automatic reloading of metadata with many identity providers
- Support for authentication contexts
- Logging for authentication events
- Customization of both SP and IDP metadata
- Processing of SAML attributes and user data using UserDetails interface
- Support for HTTP-POST, HTTP-Redirect, SOAP, PAOS and Artifact bindings
- Easy integration with applications using Spring Security
- Sample application

Internal processing of SAML messages, marshalling and unmarshalling is handled by OpenSAML [https://wiki.shibboleth.net/confluence/display/OpenSAML/Home].

You can use the following supported standards as a reference:

SAML 2.0 basic profiles

- http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
- http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf
- http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf
- http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf
- http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
- http://docs.oasis-open.org/security/saml/v2.0/saml-conformance-2.0-os.pdf

SAML 2.0 additional profiles

- http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-holder-of-key-browser-sso.pdf
- http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-idp-discovery.pdf
- http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml2-holder-of-key.pdf
- http://docs.oasis-open.org/security/saml/Post2.0/sstc-metadata-iop.pdf

eGovernment profile

- http://kantarainitiative.org/confluence/download/attachments/42139782/kantara-egov-saml2-profile-2.0.pdf

1.4 Requirements

Spring Security SAML SP requires as minimum Java 1.6.

TODO Apache Tomcat, Jetty, Oracle Weblogic, ...

1.5 Source code

Source code for the project is maintained on Github [https://github.com/SpringSource/spring-security-saml].

1.6 License

1.7 Support

Issue tracking for the module can be found at Spring Security Extensions Jira [https://jira.springsource.org/browse/SES/component/10711]. Feel free to submit bugs, patches and feature requests.

For community support please use Spring Security forum [http://forum.springsource.org/forumdisplay.php?86-SAML]. For additional support you can reach me at vladimir.schafer at gmail.com.

2. Glossary

Table 2.1. Definitions of terms used within this manual.

Term	Definition
Assertion	A part of SAML message (an XML document) which provides facts about subject of the assertion (typically about the authenticated user). Assertions can contain information about authentication, associated attributes or authorization decisions.
Artifact	Identifier which can be used to retrieve a complete SAML message from identity or service provider using a back-channel binding.
Binding	Mechanism used to deliver SAML message. Bindings are divided to front-channel bindings which use web-browser of the user for message delivery (e.g. HTTP-POST or HTTP-Redirect) and back-channel bindings where identity provider and service provider communicate directly (e.g. using SOAP calls in Artifact binding).
Discovery	Mechanism used to determine which identity provider should be used to authenticate user currently interacting with the service provider.
Metadata	Document describing one or multiple identity and service providers. Metadata typically includes entity identifier, public keys, endopoint URLs, supported bindings and profiles, and other capabilities or requirements. Exchange of metadata between identity and service providers is typically the first step for establishment of federation.
Profile	Standardized combination of protocols, assertions, bindings and processing instructions used to achieve a particular use-case such as single sign-on, single logout, discovery, artifact resolution.
Protocol	Definition of format (schema) for SAML messages used to achieve particular functionality such as requesting authentication from IDP, performing single logout or requesting attributes from IDP.
Identity provider (IDP)	Entity which knows how to authenticate users and provides information about their identity to service providers/relaying parties using federation protocols.
Service provider (SP)	Your application which communicates with identity provider in order to obtain information about the user it interacts with. User information is provided in form of security assertions.

3. Quick start guide

This chapter will guide you through steps required to easily integrate Spring Security SAML SP with SSO Circle IDP service using SAML 2.0 protocol. When done you will have a working example of Web SSO against a single Identity Provider. The steps will guide you through deployment of the module, configuration of IDP metadata (XML document describing how to connect to the IDP server using SAML 2.0 protocol) and SP metadata (XML document describing your own service) and testing of web single sign-on and single logout.

3.1 Pre-requisites

Please make sure the following items are available before starting the installation:

- Supported application server or container
- Spring Security SAML SP
- Java 1.6+ SDK
- Maven

TODO endorsed libraries

3.2 Installation steps

Compilation of the module

After downloading the Spring Security SAML SP module and unziping compile the whole project using:

```
mvn package
```

Command will create file *spring-security-saml2-sample.war* in directory *saml2-sample/target*. We will be customizing content of the application in the following steps.

Configuration of IDP metadata

Modify file *WEB-INF/classes/security/securityContext.xml* inside the war archive and replace *metadata* bean as follows:

```
<bean id="metadata" class="org.springframework.security.saml.metadata.CachingMetadataManager">
    <constructor-arg>
        <list>
            <bean class="org.opensaml.saml2.metadata.provider.HTTPMetadataProvider">
                <constructor-arg>
                    <value type="java.lang.String">http://idp.ssocircle.com/idp-meta.xml</value>
                </constructor-arg>
                <constructor-arg>
                    <value type="int">5000</value>
                </constructor-arg>
                <property name="parserPool" ref="parserPool"/>
            </bean>
        </list>
    </constructor-arg>
</bean>
```

The settings tell system to download IDP metadata from the given URL with timeout of 5 seconds. In production system metadata should be either stored as a local file or be downloaded from a source using SSL/TLS with configured trust or which provides digitally signed metadata.

Generation of SP metadata

Modify file *WEB-INF/classes/security/securityContext.xml* inside the war archive and replace *metadataGeneratorFilter* bean as follows and make sure to replace the entityId value with a string which is unique within the SSO Circle service (e.g. urn:test:yourname:yourcity):

```
<bean id="metadataGeneratorFilter" class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
    <constructor-arg>
        <bean class="org.springframework.security.saml.metadata.MetadataGenerator">
           <property name="entityId" value="replaceWithUniqueIdentifier"/>
           <property name="signMetadata" value="false"/>
        </bean>
    </constructor-arg>
</bean>
```

Deployment

Deploy the updated war archive to your application server or container. After deployment the SP module will be available at e.g. *http://localhost:8080/spring-security-saml2-sample*

Uploading of SP metadata to the IDP

Download current SP metadata:

- Open web browser at the URL of the deployed application.
- Select *Metadata information*.
- Select first item from category *Service providers*, e.g. *http://localhost:8080/spring-security-saml2-sample/saml/metadata/alias/defaultAlias*
- Copy content of the Metadata textarea to your clipboard.

Upload SP metadata to the IDP:

- Register yourself at www.ssocircle.com and login to the service.
- Select Metadata manager and click Add new Service Provider.
- Enter *entityId* configured in the section called “Generation of SP metadata” to the FQDN field.
- Paste content of clipboard to the metadata information textarea.
- Store metadata by pressing the Submit button.
- Logout from the SSOCircle service.

3.3 Testing single sign-on and single logout

Open the front page of your SP application, select *http://idp.ssocircle.com* IDP and press login. System will generate new authentication request using SAML 2.0 protocol, digitally sign it and send it to the IDP. After authentication at IDP with your account you will be redirected back to your application and automatically signed-in.

Pressing local logout will destroy local session and logout the user. Session is still active at the IDP and an attempt to reauthenticate will proceed without need to enter credentials.

Pressing global logout will destroy both local session and session at IDP.

Part II. Single sign-on with SAML

This chapter provides information about configuration and customization options of the SAML extension. It will guide you throught typical scenarios including problems you migth encounter during integration with identity providers.

4. Configuration and integration

This chapter will discuss aspects of configuring and using the library in target applications.

4.1 Overview

Spring Security SAML 2.0 library comprises three modules:

- *saml2-core* contains implementation of the WebSSO profiles of the SAML 2.0 protocol and is required for integration to target systems.
- *saml2-sample* contains example of Spring configuration used for integration to target systems. It also contains user interface for generation and management of metadata.
- *saml2-doc* contains this documentation.

Configuration of library is done using Spring context XML. An example of configuration can be found under*saml2-sample/src/main/resources/security/securityContext.xml*. Setting up of the library typically involves these steps:

- integration to application using Spring XML configuration
- configuration of signature, encryption and trust keys
- configuration of security profiles
- import, generation and customization of SP and IDP metadata
- configuration of IDP selection
- configuration of single sign-on process
- configuration of logout process
- configuration of authentication object

Additional steps such as configuration of authentication logging, customization of SAML 2.0 bindings, configuration of artifact resolution or configuration of time skews might be needed.

4.2 Integration to applications

SAML module can be directly embedded into new of existing Spring applications. In this case application itself includes the SAML library in WEB-INF/lib directory of the war archive and processes all SAML interactions. The other option of using the SAML library is deploying it as a stand-alone module which transfers information about the authenticated user to the target application using a custom mechanism. This chapter only discusses the first option.

Maven dependency

In order to include the library and all it's dependencies add the following dependency to your pom.xml file:

```
<dependency>
    <groupId>org.springframework.security.extensions</groupId>
    <artifactId>spring-security-saml2-core</artifactId>
    <version>3.1</version>
</dependency>
```

Bean definitions

Configuration of the SAML library requires beans definitions included in the *saml2-sample/src/main/resources/security/securityContext.xml* configuration file. Include copy of the file in your own Spring application, either directly or with an inclusion. Configuration steps in the following chapters will be customizing beans included in the default context.

Beans of the SAML library are using auto-wiring and annotation-based configuration by default. Make sure that your Spring configuration contains e.g. the following settings in order to enable support for these features:

```
<context:annotation-config/>
<context:component-scan base-package="org.springframework.security.saml"/>
```

Spring Security integration

Filters of the SAML module need to be enabled as part of the Spring Security settings. In case SAML authentication should be the default authentication mechanism of the application set bean *samlEntryPoint* as the default entry point. Make sure that filter *samlFilter* is included as one of the custom filters. In case SP metadata should be generated automatically during first request to the application include also filter *metadataGeneratorFilter*. The configuration directive may for example look as follows:

```
<security:http entry-point-ref="samlEntryPoint">
    <security:custom-filter before="FIRST" ref="metadataGeneratorFilter"/>
    <security:custom-filter after="BASIC_AUTH_FILTER" ref="samlFilter"/>
</security:http>
```

4.3 Metadata configuration

SAML metadata is an XML document which contains information necessary for interaction with SAML-enabled identity or service providers. Document contains e.g. URLs of endpoints, information about supported bindings, identifiers and public keys. Typically one metadata document will be generated for your own service provider and sent to all identity providers you want to enable single sign-on with. Similarly, each identity provider will make it's own metadata available for you to import into your service provider application.

Each metadata document can contain definition for one or many identity or service providers and optionally can be digitally signed. Metadata can be customized either by direct modifications to the XML document, or using extended metadata. Extended metadata is added directly to the Spring configuration file and can contain additional options which are unavailable in the basic metadata document.

Service provider metadata

Service provider metadata contains keys, services and URLs defining SAML endpoints of your application. Metadata can be either generated automatically upon first request to the service, or it can be pre-created (see Chapter 5, *Administration user interface*). Once created metadata needs to be provided to the identity providers with whom we want to establish trust.

Automatic metadata generation

Automatic metadata generation is enabled by including the following filter in the Spring Security configuration:

```
<security:custom-filter before="FIRST" ref="metadataGeneratorFilter"/>
```

Filter is automatically invoked as part of the first request to a URL processed by Spring Security. In case there is no service provider metadata already specified (meaning property *hostedSPName* of the *metadata* bean is empty) filter will generate a new one.

By default metadata will be generated with the following values which can be customized by setting properties of the *metadataGeneratorFilter* bean:

Table 4.1. Metadata generator settings

Property	**Description**	**Default value**
`entityBaseURL`	Base URL to construct SAML endpoints from, needs to be a URL with protocol, server, port and context path.	Values from the request in format: *scheme://server:port/contextPath*
`entityAlias`	Local alias for the entityId which can be part of a simple URL path and contains only alphanum characters. See Section 4.4, “Entity alias”.	*defaultAlias*
`entityId`	Unique identifier of the service provider.	<entityBaseUrl>/saml/ metadata/ alias/<entityAlias>
`requestSigned`	Flag indicating whether this service signs authentication requests.	true
`wantAssertionSigned`	Flag indicating whether this service requires signed assertions.	true
`signingKey`	Key to include with usage "signing" in the metadata. Value will be set in ExtendedMetadata as `signingKey`.	Default private key from the KeyManager
`encryptionKey`	Key to include with usage "encryption" in the metadata. Value will be set in ExtendedMetadata as `encryptionKey`.	Default private key from the KeyManager
`tlsKey`	Key to include with usage "unspecified" in the metadata. Value will be set in ExtendedMetadata as `tlsKey`.	By default not included. Key is only included in metadata when it's different from signing and encryption keys.
`signMetadata`	When true generated metadata will be signed using XML Signature using certificate with alias of `signingKey`.	true

Property	Description	Default value
`bindingsSSO`	Bindings to be included in the metadata for WebSSO profile.	POST, PAOS, HTTP-Artifact
`bindingsHoKSSO`	Bindings to be included in the metadata for WebSSO Holder-of-Key profile.	POST, HTTP-Artifact
`bindingsSLO`	Bindings to be included in the metadata for Single Logout profile.	POST, HTTP-Redirect, SOAP
`assertionConsumerIndex`	Index of assertion consumer point to be marked as default.	0
`enableDiscovery`	When true system will initialze discovery process during attempt to initialize single sign-on without pre-selected IDP.	true
`customDiscoveryURL`	When `enableDiscovery` is true value overrides default discovery request URL.	generated value for internal discovery service
`customDiscoveryResponseURL`	When `includeDiscoveryExtension` is true value overrides default discovery response URL.	generated value for entry point response URL
`includeDiscoveryExtension`	When true generated metadata will contain extension indicating that it's able to consume response from an IDP Discovery service.	false
`nameID`	Supported name identifiers to be included in the metadata.	EMAIL, TRANSIENT, PERSISTENT, UNSPECIFIED, X509_SUBJECT

By default generated metadata will be digitally signed using the default credential specified in KeyManager (see Section 4.5, “Key management” for details). Digital signature can be disabled using property `signMetadata` of the *metadataGeneratorFilter* bean.

In case application is deployed behind a reverse-proxy or other mechanism which makes the URL at the application server different from the URL seen by client at least property `entityBaseURL` should be set to a value e.g. https://www.server.com

Pre-configured metadata

TODO

Downloading metadata

TODO

Identity provider metadata

TODO

File-based metadata provider

TODO

HTTP-based metadata provider

TODO

Signature verification

TODO

Exteneded metadata

TODO

4.4 Entity alias

TODO

4.5 Key management

SAML exchanges involve usage of cryptography for signing and encryption of data. All interaction with cryptographic keys is done through interface *org.springframework.security.saml.key.KeyManager*. Default implementation relies on a single JKS key store which contains all private and public keys. KeyManager must contain at least one private key which should be marked as default by using the alias of the private key as part of the KeyManager constructor.

Make sure that your configuration of SAML module contains bean keyManager with your custom key store and passwords.

Sample keystore

Sample application contains a default key store with a sample private certificate usable for test purposes. The key store is defined as:

```
<bean id="keyManager" class="org.springframework.security.saml.key.JKSKeyManager">
    <constructor-arg value="classpath:security/samlKeystore.jks"/>
    <constructor-arg type="java.lang.String" value="nalle123"/>
    <constructor-arg>
        <map>
            <entry key="apollo" value="nalle123"/>
        </map>
```

```
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="apollo"/>
</bean>
```

First argument points to the used key store file, second contains password for the keystore, third then map with passwords for private keys with alias-password value pairs. Alias of the default certificate is the last parameter.

Generating and importing private keys

TODO

Importing public keys

TODO

Loading SSL/TLS certificates

TODO

4.6 Security profiles

Exchanges of messages between identity providers and service providers with SAML protocol involves usage of digital signatures. Signatures are typically constructed using means of asymetric cryptography and public key infrastructure with public and private keys signed by trusted certification authorities. Signatures are either applied directly to parts of XML representation of SAML messages using XML Signature or are part of the transport layer used to deliver the message like SSL/TLS.

Verification of signatures is executed in two phases. Signature is first checked for validity by comparing digital hash included as part of the signature with value calculated from the content. Subsequently it is verified whether party who created the signature is trusted by the recipient. Module provides two mechanisms for defining which signatures should be accepted - metadata interoperability mode and PKIX mode.

Security profiles are defined in Extended Metadata of your local SP. Profile can be defined separately for XML Signatures using property *securityProfile* and for SSL/TLS Signatures using property*sslSecurityProfile*. Value of both properties can be either *metaiop* or *pkix*. For details about using Extended Metadata see Section 4.3, “Metadata configuration”, for reference of allowed values see Section A.1, “Extended metadata”.

Metadata interoperability profile (MetaIOP)

With MetaIOP mode certificates are not checked for expiration or revocation and certificate paths are not verified. This means that it does not matter which certification authority issued the certificate, as the fact whether the certificate is trusted or not is conveyed using other mechanisms (e.g. by secure metadata exchange or digital signature of metadata itself).

Signature is deemed trusted when the certificate used to create it is included in one of the following places:

- Key with usage of signing or unspecified in entity metadata of a remote entity
- Signing key specified in property *signingKey* of extended metadata of a remote entity

MetaIOP is the default profile for verification of XML signatures. For details about this profile see the specification [http://docs.oasis-open.org/security/saml/Post2.0/sstc-metadata-iop.pdf].

PKIX profile

With PKIX profile trust of signature certificates is verified based on a certificate path between trusted CA certificates and the certificate in question. Certificate is trusted when it's possible to construct path from a trusted certificate to the validated one. With this profile certificate expiration and revocation can be checked.

Trusted keys (anchors) for PKIX verification of signatures are combined from the following places:

- Key with usage of signing or unspecified in entity metadata of a remote entity
- Signing key specified in property *signingKey* of extended metadata of a remote entity
- All keys specified in *trustedKeys* set of extended metadata of a remote entity

Custom profile

Engine used to verify trust of signatures for given combination of SP/IDP is created in methods *populateTrustEngine* and *populateSSLTrustEngine* of interface *org.springframework.security.saml.context.SAMLContextProvider* and can be overriden with custom implementation. See Section 4.12, “Context provider” for details on context customization.

4.7 IDP selection

TODO

4.8 Single sign-on process

TODO

4.9 Logout process

TODO

4.10 Authentication object

TODO

4.11 Authentication log

TODO

4.12 Context provider

TODO

5. Administration user interface

TODO

6. Troubleshooting

Appendix A. Configuration reference

This chapter provides reference for settings available in configuration beans of the SAML module.

A.1 Extended metadata

Extended metadata provides additional settings for customization of IDP and SP behavior. Bean can be found in package*org.springframework.security.saml.metadata.ExtendedMetadata*. For details on setting up metadata please consultSection 4.3, “Metadata configuration”.

Table A.1. Extended metadata settings

Property	Default	Entities	Description
local	false	both	
alias		local	
idpDiscoveryEnabled	false	local	
idpDiscoveryURL		local	
idpDiscoveryResponseURL		local	
ecpEnabled	false	local	
securityProfile	metaiop	local	
sslSecurityProfile	pkix	local	
signingKey		both	
encryptionKey		both	
tlsKey		both	
trustedKeys		both	
requireLogoutRequestSigned		both	
requireLogoutResponseSigned		both	
requireArtifactResolveSigned		both	

A.2 WebSSO profile options

Configuration bean can be found in *org.springframework.security.saml.websso.WebSSOProfileOptions* and provides means to customize authentication request sent to the IDP. For details on setting the login process please see Section 4.8, “Single sign-on process”.