ISBN: 978-89-294-1148-0-95560

SAML 및 OIDC 인증규약을 활용한 웹 응용서비스의 통합인증 구현방법

- Korean Access Federation (KAFE) -

https://www.kafe.or.kr

조진용 (jiny92@kisti.re.kr)

주의 사항

· 본 문서의 내용을 충분히 숙지한 후에 환경구축을 시작하시기 바랍니다.

· 본 문서의 이용 대상은 다음과 같습니다.
  - 이용 대상자: 통합인증 관련 개발자
  - 대상 서비스: 웹 응용서비스

· 개발/운용환경 구축에 본 문서의 특정 내용을 “복사하여 붙여넣기”를 하지 마십시오. 오류의 원인이 됩니다.

· 문서에 포함된 일부 변수(비밀번호, FQDN 등)들을 구축하는 환경에 맞게 활용하시기 바랍니다.

목 차

Chapter 1. Introduction

Chapter 2. SAML Integration

Chapter 3. 계정연합 참여

용어 정의

연합인증 · 통합인증(Single Sign On)의 확장된 개념이다. 표준 인증규약(SAML)을 이용한다.

계정연합 · Identity(ID) Federation, 서비스제공자와 식별정보제공자의 연합으로써 동일한 연합인증 정책을 준용한다.

KAFE · Korean Access Federation, 국내 계정연합으로써 국가과학기술연구망(KREONET)에서 운영한다.

식별정보제공자 · Identity Provider, 사용자에게 로그인 기능을 제공하고 사용자를 인증하는 SAML 개체로써 일반적으로 기관과 조직을 의미한다.

서비스제공자 · Service Provider, 정보시스템(웹 응용 등)을 제공하는 SAML 개체로써 일반적으로 웹 기반 응용서비스를 의미한다.

탐색서비스 · 식별정보제공자를 선택할 수 있는 서비스이다. 연합인증에는 다수의 식별정보제공자(기관)이 포함되기 때문에 사용자는 하나의 식별정보제공자를 선택해야 한다.

개체식별자 · entityId, 서비스제공자 또는 식별정보제공자를 가리키는 고유식별자이다. https://FQDN/idp/simplesaml 등과 같은 방법으로 표기한다.

메타데이터 · SAML 개체의 앤드포인트, 인증서 등의 정보를 담고 있는 XML 파일이다. 식별정보제공자와 서비스제공자는 메타데이터를 교환해야 서로 통신이 가능하다.

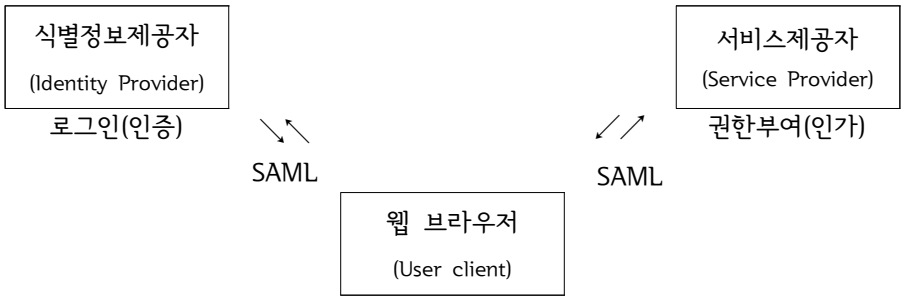
속성 · Attribute, 사용자가 갖는 정보(예, email, 이름 등)를 의미한다.

Chapter 1. Introduction

1. 개요

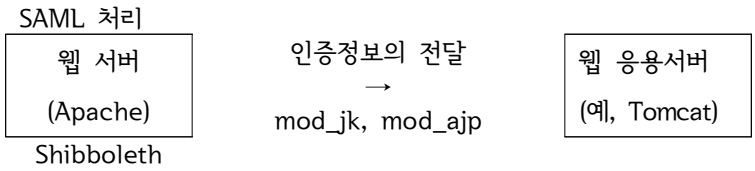
본 문서는 웹 응용에 SAML(Security Assertion Markup Language) 소프트웨어를 통합하는 방법을 기술한다. SAML은 XML 기반의 국제표준 보안인증 규격으로써 서비스제공자와 식별정보제공자는 HTTP를 이용해 SAML 메시지를 교환한다.

연합인증에서 서비스제공자와 식별정보제공자는 사용자의 웹 브라우저를 이용해 SAML 메시지를 교환한다. 서비스제공자에서 식별정보제공자에게 로그인을 요청할 때는 HTTP-Redirect, 식별정보제공자가 서비스제공자에게 로그인 결과를 전달할 때는 HTTP-POST를 사용하는 것이 일반적이다. 서비스제공자는 HTTP-POST 방식으로 전달받은 인증정보(예, 언제 누가 로그인했으며 로그인결과는 성공)와 사용자속성정보(예, 이메일, 이름, 직무정보 등)을 이용해 해당 사용자를 인가한다.



연합인증에서 주로 사용하는 SAML 소프트웨어와 특징은 다음과 같다.

Shibboleth	Apache 모듈로써 Web 서버와 Web 응용 서버가 분리된 경우 이용
simpleSAMLphp	PHP 라이브러리로 웹 응용이 PHP로 구현된 경우 이용
Spring Security SAML	Spring 프레임워크를 이용하는 웹 응용의 경우 이용



Shibboleth를 이용할 경우, 연합인증과 관련된 기능은 모두 웹 서버의 Shibboleth 모듈(mod_shib)에서 처리한다. Shibboleth가 획득한 인증정보와 속성정보는 mod_jk 또는 mod_ajp를 이용해 웹 응용서버에 전달되고 웹 응용서버는 전달받은 속성정보를 이용해 사용자를 인가(Authorization)한다. 웹 응용서버가 JEUS일 경우에는 mod_jk를 이용해 속성정보를 전달해야 한다.

Shibboleth를 이용할 수 없는 환경(예, 웹 서버가 WebtoB) 또는 웹 서버가 존재하지 않는 환경이라면 웹 응용서버에 설치될 웹 응용에 SAML 라이브러리를 통합함으로써 로그인 메시지를 처리케 해야 한다. 웹 서버가 WebtoB이고 웹 응용서버가 JEUS라면 Java 기반의 SAML Library를 이용해 웹 응용을 구현해야 한다.

본 문서는 웹 응용서버(Tomcat이나 JEUS)에서 구동되고 전자정부프레임워크를 이용해 개발된 웹 응용을 대상으로 Spring Security SAML extension을 통합하는 방법을 기술한다. 또한 Shibboleth를 이용해 SAML 기능을 활성화시키는 방법에 대해서도 부가적으로 설명한다. Spring Security SAML extension의 매뉴얼은 아래 링크에서 찾을 수 있다.

- https://docs.spring.io/spring-security-saml/docs/current/reference/html

2. 기술 프로파일

2.1 속성 정보

서비스제공자는 식별정보제공자가 전달한 속성정보를 이용해 사용자를 인가(접근권한의 부여)해야 한다. 이용 가능한 속성정보는 https://www.kafe.or.kr/attributeMap에서 확인할 수 있다. 서비스제공자는 OID 형태의 속성정보를 처리해야 한다.

아래 표는 KAFE에서 활용 중인 속성정보의 일부를 보여준다. 일반적으로 식별정보제공자가 uid를 제공하지 않기 때문에 서비스제공자는 eduPersonTargetedID(eptid)나 eduPersonPrincipalName(eppn)을 사용자에 대한 고유식별자로 활용해야 한다. 한명의 사용자가 다수의 email을 가질 수 있으므로 email은 사용자의 고유식별자로 적합하지 않다. 하나의 속성은 다수의 값(multiple value)을 갖거나 단일 값(single value)을 가질 수 있으므로 구현 시 반영해야 한다.

속성명	oid	설명
eduPersonTargetedID	1.3.6.1.4.1.5923.1.1.1.10	고유식별자
eduPersonPrincipalName	1.3.6.1.4.1.5923.1.1.1.6	고유식별자
sn	2.5.4.4	성
givenName	2.5.4.42	이름
cn	2.5.4.3	성명
displayName	1.3.18.0.2.4.715	화면표시 이름
mail	0.9.2342.19200300.100.1.3	이메일
eduPersonAffiliation	1.3.6.1.4.1.5923.1.1.1.1	직무
eduPersonScopedAffiliation	1.3.6.1.4.1.5923.1.1.1.9	기관내 직무정보
organizationName	2.5.4.10	기관명

eduPersonTargetedID는 특정 서비스제공자에서만 유효한 고유식별자이다. 예를 들어, 사용자 gildong hong의 eduPersonTargetedID는 서비스제공자 #1에서 fed01aaef204ae이고 서비스제공자 #2에서 02deffecc304ff일 수 있다. eduPersonPrincipalName은 모든 서비스제공자에서 유일한(unique) 고유식별자이다. 예를 들어, 사용자 gildong hong의 eduPersonPrincipalName이 서비스제공자 #1에서 fed01aaef204ae@kafe.or.kr이라면 서비스제공자 #2에서도 fed01aaef204ae@kafe.or.kr이다.

자체적으로 로그인 기능을 가지고 있는 서비스제공자가 추가적으로 SAML을 이용한 사용자 로그인을 허용하고자 한다면 로컬로그인 계정과 SAML을 통해 로그인되는 계정을 상호연결(ID 연결서비스라 칭함)해야 한다. ID 연결서비스가 필요한 서비스제공자는 SAML을 통해 로그인되는 사용자 계정의 고유식별자로 eduPersonPrincipalName을 이용해야 한다. 예를 들어, 식별정보제공자가 전달한 eduPersonPrincipalName의 값과 로컬계정의 uid를 서로 연결시킴으로써 SAML로 로그인한 사용자와 로컬 사용자를 맵핑할 수 있다.

SAML 속성정보	Local 계정정보	ID 연결
· eppn={a0bec@home.org} · name = {gildong hong}	· uid = {gdhong} · name = {gildong hong} · email = {hong@home.org} …,	if eppn=a0bec@home.org then; the user is gdhong fi

SAML 전달정보		로컬 계정DB
		uid
		password
		전화번호
		주소
		과학기술인등록번호
		기관코드
eduPersonPrincipalName	→	eduPersonPrincipalName
email	→	email
displayName	→	displayName

서비스제공자가 필요로 하는 속성정보는 서비스제공자의 메타데이터에 기록되어 식별정보제공자에게 배포된다. 식별정보제공자는 서비스제공자가 요구하는 속성정보 중, 보유한 속성정보만 제공한다.

서비스제공자가 사용자 인가를 위해 필요한 속성정보를 전달받지 못했다면 서비스제공자는 해당 사용자를 예외처리를 해야 한다. 예를 들어, 필요하지만 전달받지 못한 속성정보를 사용자가 입력할 수 있도록 추가정보 입력화면을 제공하거나 오류를 발생시켜야 한다.

Chapter 2. SAML Integration

1. Spring SAML for Java Web Applications

1.1 검증 환경

Spring Security SAML Extension은 Java 1.6 이상을 필요로 한다. 본 문서의 내용은 다음 표와 같은 환경에서 검증되었다.

	상세
검증 환경	JDK 1.8.0
	Tomcat 8.x 버전 (Tomcat 9에서는 구동 안됨)
	Maven

이용된 예제 코드는 아래 주소에서 내려 받을 수 있다. git 또는 zip 파일을 내려 받은 후 IDE(Integrated Development Environment)에서 import 기능을 사용하여 프로젝트를 불러온다.

	상세
Spring 예제 저장소	https://git.kreonet.net/kafe-support/spring-security-saml2-sample
Spring 예제 다운로드	https://git.kreonet.net/kafe-support/spring-security-saml2-sample/-/archive/master/spring-security-saml2-sample-master.tar.gz

1.2 메타데이터의 구성

서비스제공자는 메타데이터를 생성하고 식별정보제공자와 메타데이터를 교환해야 한다.

1.2.1 서비스제공자의 메타데이터 설정

서비스제공자의 메타데이터를 설정하기 위해서 WEB-INF 폴더의 securityContext.xml 파일을 수정한다.

코드 1-1 WEB-INF/securityContext.xml

```
<bean id="metadataGeneratorFilter"
  class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
  <constructor-arg>
    <bean
      class="org.springframework.security.saml.metadata.MetadataGenerator">
      <property name="entityId" value="http://localhost:8080/sp/spring" />
      <property name="extendedMetadata">
        <bean
          class="org.springframework.security.saml.metadata.ExtendedMetadata">
          <property name="signMetadata" value="false" />
          <property name="idpDiscoveryEnabled" value="true" />
```

```
        </bean>
      </property>
    </bean>
  </constructor-arg>
</bean>
```

중요

개체식별자 및 URL 설정 시 주의 사항

· 반드시 https로 시작해야 함(https를 지원해야 함)
· 80(HTTP)이나 443(HTTPS)를 이용할 경우, 포트번호가 노출되지 않도록 설정

- 서비스제공자의 개체식별자(entityId) 설정

개체식별자는 식별정보제공자 또는 서비스제공자를 나타내는 고유식별자이다. 계정연합 내에서 개체식별자는 중복될 수 없다. Spring 기반의 서비스제공자인 경우, KAFE는 다음과 같은 표기 규정을 권고하고 있다. 필요할 경우, 코드 1-1의 http://localhost:8080/sp/spring을 적절히 수정한다.

https://FQDN/[entity type]/[saml software or framework]

- signMetadata 속성의 value 값은 메타데이터를 서명할 것인지의 여부를 의미한다. 기본값은 false이다.
- idpDiscoveryEnabled 값은 식별정보제공자 탐색서비스의 이용 여부를 의미한다. 계정연합에 참여하고 식별정보제공자와 서비스제공자가 N:1로 연동될 때는 탐색서비스를 이용해야 한다. 탐색서비스를 이용할 경우 value는 true로 설정한다.

중요

개체식별자 설정 예시

· Shibboleth 기반 서비스제공자: https://FQDN/sp/shibboleth
· Simplesamlphp 기반 서비스제공자: https://FQDN/sp/simplesamlphp
· 전자정부/Spring 기반 서비스제공자: https://FQDN/sp/spring

※ FQDN은 서버의 도메인 이름으로 DNS에 등록되어 있어야 함

1.2.2 자가서명 인증서(Self-signed certificate)의 설치 및 설정

Tip

자가서명 인증서

SAML 메타데이터는 공개키 기반의 인증서 정보를 포함해야 한다. 인증서는 메시지의 복호화와 전자서명의 검증 등에 활용된다.

keystore 파일(samlKeystrore.jks)이 있는 디렉토리로 이동한 후 keytool을 이용해 자가서명 인증서(코드 1-2 참조)를 생성한다.

※ 자가서명 인증서는 HTTPS 서비스 제공을 위한 SSL 인증서와 구분해 이용해야 한다.

keystore 파일 위치	spring-security-saml-master/spring-security-saml-master/sample/src/main/resources/security/

코드 1-2	keytool 명령어 명세
keytool -genkey -alias [인증서 이름] -validity 3650 -keyalg RSA -sigalg SHA256withRSA -keysize 2048 -keystore samlKeystore.jks -keypass [인증서 암호] -storepass [keystore 암호] -dname "CN=[도메인명 or IP 주소],OU=[부서명],O=[조직명],L=[위치],S=[도시],C=[국가명]"	

코드 1-3	Linux 예제
$ keytool -genkey -alias apollo -validity 3650 -keyalg RSA -sigalg SHA256withRSA -keysize 2048 -keystore samlKeystore.jks -keypass yourpassword -storepass nalle123 -dname "CN=localhost,OU=KAFE,O=KISTI,L=Yuseonggu,S=Daejeon,C=KR"	

코드 1-4	Windows 예제
$ "C:\Program Files\Java\jdk1.8.0_131\jre\bin\keytool.exe" -genkey -alias apollo -validity 3650 -keyalg RSA -sigalg SHA256withRSA -keysize 2048 -keystore samlKeystore.jks -keypass yourpassword -storepass nalle123 -dname "CN=localhost,OU=KAFE,O=KISTI,L=Yuseonggu,,S=Daejeon,C=KR"	

※ 위 붉은색으로 표시된 부분은 서비스 환경에 맞도록 수정해야 함

중요

자가서명 인증서 생성 규정

· CN=localhost의 'localhost'는 서버의 실제 FQDN(도메인명. 예, ex.kisti.re.kr)과 일치해야 함

· -sigalg 알고리즘은 SHA256withRSA, -keysize는 2048 또는 3072, -dname의 세부항목은 서비스 제공자의 세부 환경에 맞춰 정확히 입력해야 함

아래 명령어를 통해 자가서명 인증서를 검증 할 수 있다. keystore 이름과 keystore 암호를 keyManager에서 확인할 수 있다. 예시에서는 인증서 파일 이름은 samlKeystore.jks, 인증서 alias는 apollo, keystore 암호(-storepass)는 nalle123 이 이용되었다.

코드 1-5	Linux
$ keytool -list -keystore samlKeystore.jks -v -alias apollo	

코드 1-6	Windows
$ "C:\Program Files\Java\jdk1.8.0_131\jre\bin\keytool.exe" -list -keystore samlKeystore.jks -v -alias apollo	

생성한 인증서 정보를 Spring Security SAML 설정에 반영하기 위해 아래 코드 1-7의 적색부분을 수정한다.

코드 1-7	WEB-INF/securityContext.xml

```
<bean id="keyManager"
  class="org.springframework.security.saml.key.JKSKeyManager">
  <constructor-arg value="classpath:security/[jks_파일명]" />
  <constructor-arg type="java.lang.String" value="[storepass]" />
  <constructor-arg>
    <map>
      <entry key="[alias명]" value="[storepass]" />
    </map>
  </constructor-arg>
  <constructor-arg type="java.lang.String" value="[alias명]" />
</bean>
```

- jks_파일명: eg. samlKeystore.jks
- alias: eg. apollo
- storepass: eg. nalle123

※ 예제에서 사용한 값 대신 인증서를 생성할 때 사용한 정보를 바탕으로 입력

1.3 메타데이터의 교환

메타데이터 생성을 위한 securityContext 설정이 완료되면 로그인을 수행할 식별정보제공자의 메타데이터를 서비스제공자에 등록해야 한다. 구현중인 서비스제공자와 KAFE에서 제공하는 시험용 식별정보제공자(https://testidp.kreonet.net)의 서로 교환한다. 시험용 식별정보제공자의 메타데이터는 https://testidp.kreonet.net/simplesaml/saml2/idp/metadata.php에서 얻을 수 있다.

1.3.1. 시험용 식별정보제공자의 메타데이터를 서비스제공자에 등록

구현중인 서비스제공자의 securityContext.xml(WEB-INF 폴더) 파일에 시험용 식별정보제공자의 메타데이터를 등록한다. https://git.kreonet.net/에서 다운받은 예제 코드에는 시험용 식별정보제공자의 메타데이터가 이미 설정되어 있다.

코드 1-8	WEB-INF/securityContext.xml

```
<bean id="metadata" class="org.springframework.security.saml.metadata.CachingMetadataManager">
  <constructor-arg>
    <list>
      <bean class="org.springframework.security.saml.metadata.ExtendedMetadataDelegate">
        <constructor-arg>
          <bean class="org.opensaml.saml2.metadata.provider.HTTPMetadataProvider">
            <constructor-arg>
              <value type="java.lang.String">
                https://testidp.kreonet.net/simplesaml/saml2/idp/metadata.php
              </value>
            </constructor-arg>
            <constructor-arg>
              <value type="int">5000</value>
            </constructor-arg>
            <property name="parserPool" ref="parserPool" />
          </bean>
        </constructor-arg>
        <constructor-arg>
          <bean class="org.springframework.security.saml.metadata.ExtendedMetadata"></bean>
        </constructor-arg>
        <property name="metadataTrustCheck" value="false" />
      </bean>
    </list>
  </constructor-arg>
</bean>
```

코드 1-8에 따라 정상적으로 설정했으나 메타데이터 획득에 실패(오류가 발생)하면 시험용 식별정보제공자의 메타데이터를 파일로 내려받아(코드 1-9 참조) 서비스제공자의 securityContext.xml에 추가(코드 1-10 참조)해야 한다.

코드 1-9	시험용 ID 제공자 메타데이터 다운로드 명령어

```
$ cd src/main/resources/metadata

$ wget https://testidp.kreonet.net/simplesaml/saml2/idp/metadata.php -O testidp.xml
```

코드 1-10	WEB-INF/securityContext.xml

```
<bean class="org.springframework.security.saml.metadata.ExtendedMetadataDelegate">
  <constructor-arg>
    <bean class="org.opensaml.saml2.metadata.provider.ResourceBackedMetadataProvider">
      <constructor-arg>
        <bean class="java.util.Timer" />
      </constructor-arg>
      <constructor-arg>
        <bean class="org.opensaml.util.resource.ClasspathResource">
          <constructor-arg value="/metadata/testidp.xml" />
        </bean>
      </constructor-arg>
      <property name="parserPool" ref="parserPool" />
    </bean>
  </constructor-arg>
  <constructor-arg>
    <bean class="org.springframework.security.saml.metadata.ExtendedMetadata"></bean>
  </constructor-arg>
  <property name="metadataTrustCheck" value="false" />
</bean>
```

1.3.2. 시험용 식별정보제공자에 서비스제공자의 메타데이터 등록

서비스제공자에 시험용 식별정보제공자의 메타데이터가 등록되었으므로 이번에는 시험용 식별정보제공자에 서비스제공자의 메타데이터를 등록해야 한다. 예제 프로젝트를 빌드한 후, 아래 링크를 통해 서비스제공자의 메타데이터를 다운로드한다. 아래 붉은색으로 표시된 부분은 구현환경에 맞춰 사용한다.

메타데이터 주소	http://localhost:8080/spring-security-saml2-sample/saml/metadata

```
            </ds:KeyInfo>
        </md:KeyDescriptor>
        <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="http://localhost:8080/spring-security-saml2-samp
        <md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect" Location="http://localhost:8080/spring-security-saml2
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</md:NameIDFormat>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</md:NameIDFormat>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</md:NameIDFormat>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</md:NameIDFormat>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName</md:NameIDFormat>
        <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="http://localhost:8080/spring-security-saml
        <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact" Location="http://localhost:8080/spring-security-

        <md:AttributeConsumingService index="2">
            <md:ServiceName xmlns:xml="http://www.w3.org/XML/1998/namespace" xml:lang="en"></md:ServiceName>
            <md:RequestedAttribute FriendlyName="email" Name="urn:oid:0.9.2342.19200300.100.1.3" NameFormat="urn:oasis:names:tc:SAML:2.0:attrn
            <md:RequestedAttribute FriendlyName="uid" Name="urn:oid:0.9.2342.19200300.100.1.1" NameFormat="urn:oasis:names:tc:SAML:2.0:attrnam
            <md:RequestedAttribute FriendlyName="displayName" Name="urn:oid:2.16.840.1.113730.3.1.241" NameFormat="urn:oasis:names:tc:SAML:2.0
            <md:RequestedAttribute FriendlyName="eduPersonTargetedID" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" NameFormat="urn:oasis:names:tc:
            <md:RequestedAttribute FriendlyName="eduPersonPrincipalName" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" NameFormat="urn:oasis:names:t
            <md:RequestedAttribute FriendlyName="organizationName" Name="urn:oid:2.5.4.10" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-fo
            <md:RequestedAttribute FriendlyName="schacHomeOrganization" Name="urn:oid:1.3.6.1.4.1.25178.1.2.9" NameFormat="urn:oasis:names:tc:
            <md:RequestedAttribute FriendlyName="eduPersonAffiliation" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1" NameFormat="urn:oasis:names:tc:
        </md:AttributeConsumingService>
    </md:SPSSODescriptor>
</md:EntityDescriptor>
```

<서비스제공자의 메타데이터(예시)>

<서비스제공자의 메타데이터(예시)>는 다운로드 받은 서비스제공자의 메타데이터 파일 중 일부를 보여준다. Spring Security SAML에서 생성되는 메타데이터에는 서비스제공자가 필요로 하는 속성정보(사용자 정보, 위 그림은 파란색 부분)가 포함되지 않는다. 서비스제공자가 필요로 하는 속성정보를 메타데이터에 추가하기 위해서 위 그림의 파란색 부분처럼 코드 1-11의 내용을 추가한다. </md:SPSSODescriptor> 앞에 코드 1-11의 내용을 복사해 붙여 넣는다.

※ 식별정보제공자는 서비스제공자의 메타데이터에 포함된 속성명을 참조해서 속성정보를 제공한다. 예를 들어, 서비스제공자의 메타데이터에 요구속성이 <md:RequestedAttribute FriendlyName="email" Name="urn:oid:0.9.2342.19200300.100.1.3" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/>만 포함되어 있다면 식별정보제공자는 로그인한 사용자의 이메일 주소만 전달한다.

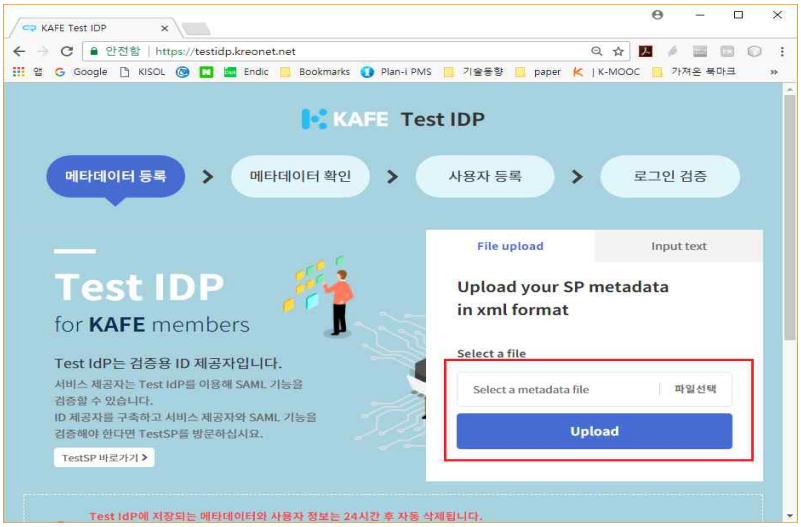
코드 1-11	Metadata의 속성정보

```
<md:AttributeConsumingService index="2"><md:ServiceName
xmlns:xml="http://www.w3.org/XML/1998/namespace"
xml:lang="en"></md:ServiceName><md:RequestedAttribute FriendlyName="email"
Name="urn:oid:0.9.2342.19200300.100.1.3" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
isRequired="true"/><md:RequestedAttribute FriendlyName="uid" Name="urn:oid:0.9.2342.19200300.100.1.1"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/><md:RequestedAttribute
FriendlyName="displayName" Name="urn:oid:2.16.840.1.113730.3.1.241"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/><md:RequestedAttribute
FriendlyName="eduPersonTargetedID" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/><md:RequestedAttribute
FriendlyName="eduPersonPrincipalName" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/><md:RequestedAttribute
FriendlyName="organizationName" Name="urn:oid:2.5.4.10"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/><md:RequestedAttribute
FriendlyName="schacHomeOrganization" Name="urn:oid:1.3.6.1.4.1.25178.1.2.9"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/><md:RequestedAttribute
FriendlyName="eduPersonAffiliation" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
isRequired="false"/></md:AttributeConsumingService>
```

KAFE에서 이용하는 사용자 속성정보의 일부는 다음 표와 같다. 이용할 수 있는 속성정보의 전체 목록은 https://www.kafe.or.kr/attributeMap를 참조한다. 서비스제공자가 요청하는 속성정보 중 식별정보제공자가 제공할 수 없는 속성정보는 서비스제공자에게 전달되지 않는다. 예를 들어, 서비스제공자가 주민등록번호를 요청했어도 식별정보제공자가 로그인한 사용자의 주민등록번호를 제공할 수 없으면 주민등록번호는 서비스제공자에게 전달되지 않는다.

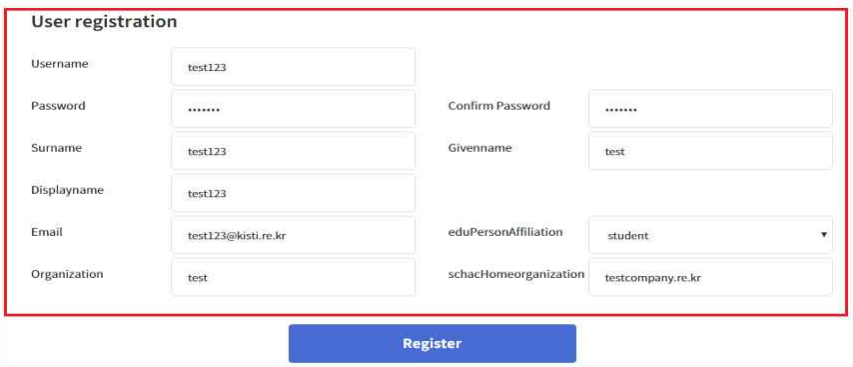
속성명	oid	설명
eduPersonTargetedID	1.3.6.1.4.1.5923.1.1.1.10	고유식별자
eduPersonPrincipalName	1.3.6.1.4.1.5923.1.1.1.6	고유식별자
sn	2.5.4.4	성
givenName	2.5.4.42	이름
cn	2.5.4.3	성명
displayName	1.3.18.0.2.4.715	화면표시 이름
mail	0.9.2342.19200300.100.1.3	이메일
eduPersonAffiliation	1.3.6.1.4.1.5923.1.1.1.1	직무
eduPersonScopedAffiliation	1.3.6.1.4.1.5923.1.1.1.9	기관내 직무정보
organizationName	2.5.4.10	기관명

서비스제공자의 메타데이터가 확보되면 시험용 식별정보제공자(https://testidp.kreonet.net)에 접속해 메타데이터를 등록한다. <시험용 식별정보제공자>는 KAFE에서 제공하는 시험용 식별정보제공자를 보여준다.



<시험용 식별정보제공자>

그림 <시험용 식별정보제공자>의 붉은색 상자로 표시된 부분에서 '파일선택'을 클릭해 서비스제공자의 메타데이터 파일을 선택하고 'Upload' 버튼을 클릭하면 서비스제공자의 메타데이터가 시험용 식별정보제공자에 등록된다. 'Input text' 탭을 클릭하면 서비스제공자의 메타데이터를 copy-and-paste 할 수 있다.



<시험용 식별정보제공자에서 사용자 등록>

시험용 식별정보제공자의 '사용자 등록' 메뉴를 클릭하면 그림 <시험용 식별정보제공자에서 사용자 등록>과 같이 화면이 나타난다. 해당 식별정보제공자에서 로그인이 가능한 임시 사용자를 생성할 수 있다. 'Username' 등은 임의로 입력해도 되지만 실제 이름 등 개인정보를 입력하지 않도록 주의한다. 'Username'은 사용자 ID를 의미한다. 등록된 서비스제공자의 메타데이터와 사용자계정은 24시간 동안 유효하다.

1.3.3. 시험용 시험용 식별정보제공자와 서비스제공자의 연동 확인

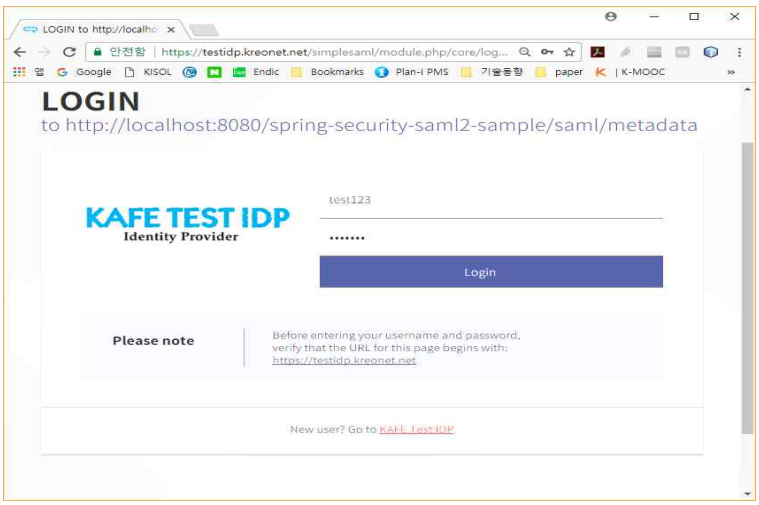
예제 프로젝트를 빌드한 후, 웹 브라우저를 이용해 예제 응용서비스에 접근한다. 붉게 표시된 부분은 서비스제공자의 구축환경에 맞춰 적절히 수정한다.

예제 프로그램의 접속 주소	http://localhost:8080/spring-security-saml2-sample/

SAML Login을 클릭하면, 그림 <예제 탐색서비스>와 같은 내장형 탐색서비스를 확인할 수 있다. 나타나는 식별정보제공자(예, https://testidp.kreonet.net/idp/simplesaml)의 목록은 차이가 있을 수 있다. 또한, KAFE에서 제공하는 중앙형 탐색서비스를 이용하면 아래 보이는 것과 다른 화면이 나타난다.

<예제 탐색서비스>

시험용 식별정보제공자를 이용하기 위해서 https://testidp.kreonet.net/idp/simplesamlphp를 선택한다. 'Start Single Sign On'을 클릭하면 그림 <로그인 화면>처럼 시험용 서비스제공자의 로그인 화면이 나타난다.



<로그인 화면>

1.3.2절에서 생성한 사용자 ID(Username)과 비밀번호를 입력하고 로그인을 시도한다.

isLogin	true
urn:oid:1.3.6.1.4.1.25178.1.2.9	kisti.re.kr
urn:oid:0.9.2342.19200300.100.1.3	proin@kisti.re.kr
urn:oid:1.3.6.1.4.1.5923.1.1.1.1	staff
urn:oid:0.9.2342.19200300.100.1.1	proin
urn:oid:2.16.840.1.113730.3.1.241	Yeonghun Chae
urn:oid:1.3.6.1.4.1.5923.1.1.1.6	G/A2s/7yYYBSd3U1hmV0sgu0kBI=@kreonet.net
urn:oid:2.5.4.10	KISTI
_springSamlStorageKey	{}
SPRING_SECURITY_CONTEXT	org.springframework.security.core.context.SecurityContextImpl@69418269: Authentication: org.springframework.security.providers.ExpiringUsernameAuthenticationToken@69418269: Principal: 550686acc89a6d6b6f94424fa74f497764c9f98f; Credentials: [PROTECTED]; Authenticated: true; Details: null; Not granted any authorities

<속성정보의 화면표시>

로그인에 성공하면, 시험용 식별정보제공자가 전달한 사용자 속성정보를 서비스제공자에서 확인할 수 있다(위 <속성정보의 화면표시>를 참조).

1.4 웹 응용과 Spring Security SAML 통합을 위한 상세 기술

본 절에서는 1.3절의 내용을 보다 구체적으로 설명하고 실제 웹 응용에 Spring Security SAML의 적용 시 고려해야 할 기술적 사항에 대해서 살펴본다.

Tip

개발환경에서 도메인 이름과 HTTPS 설정

- 개발환경에서는 도메인이름을 hosts 파일에 임의 설정하거나 도메인이름 대신에 IP 주소 이용 가능
- 개발환경에서는 HTTP를 이용하거나 Self-signed 인증서를 이용해 HTTPS를 이용 가능
  ※ Self-signed 인증서 대신에 letsencrypt(https://letsencrypt.org) 등 무료 SSL/TLS를 이용 가능

1.4.1. 예제 코드/프로젝트의 구조

https://git.kreonet.net에서 내려 받은 예제 코드는 다음과 같은 구조를 가지고 있다. 아래 구조를 참조하여 기 운영 중인 웹 응용서비스에 Spring Security SAML을 통합한다.

sample-project/

- src/main/java/
  - org.springframework.security.saml.web **(웹 응용에 통합 시, 붉은색 파일들의 복사 필요)**
    - MetadataController.java
    - MetadataForm.java
    - MetadataValidator.java
- src/main/resources/
  - metadata/idp.xml **(웹 응용에 통합 시, 파일 다운로드 또는 신규 생성 필요)**
  - security/samlKeystore.jks **(웹 응용에 통합 시, 신규 생성 필요)**
- src/main/webapp/
  - index.jsp
  - logout.jsp
  - samlhandler.jsp
  - session.jsp
  - WEB-INF/ **(웹 응용에 통합 시, 복사 후 웹 응용의 환경에 맞게 설정 변경)**
    - securityContext.xml **(기존 웹 응용의 설정과 비교하여 필요한 부분 수정 및 추가)**
    - security/idpSelection.jsp **(탐색서비스를 직접 구현하는 경우, UI 수정 필요)**

예제 코드가 웹 응용에 정상적으로 통합되면 아래 표의 링크를 통해 로그인, 로그아웃 등의 기능을 검증 할 수 있다. 링크주소는 구축된 환경에 따라 달라질 수 있다. 연합인증에서 식별정보제공자는 로그인에 성공한 사용자의 세션정보를 특정 시간만큼 유지한다.

SAML 로그인	http://your-service-domain/saml/login
SAML 로그아웃(로컬)	http://your-service-domain/saml/logout?local=true
SAML 로그아웃(SLO)	http://your-service-domain/saml/logout
메타데이터 내려받기	http://your-service-domain/saml/metadata

Tip

싱글로그아웃(SLO)과 로그아웃(로컬)의 차이

· 싱글로그아웃(SLO) 요청의 처리는 식별정보제공자가 담당한다. 사용자는 로그인한 모든 서비스에서 로그아웃할 수 있다.

· 반면에 로그아웃(로컬)은 식별정보제공자에게 로그아웃을 요청하지 않고 브라우저 세션만 삭제한다.

1.4.2. securityContext.xml

이미 운용 중인 Spring 기반의 웹 응용에 Spring Security SAML을 통합할 경우, 운용 중인 웹 응용과 Spring Security SAML의 Context가 충돌되지 않도록 설계해야 한다. Spring에서 인증·인가의 처리는 https://minwan1.github.io/2017/04/22/2017-04-22-spring-security-implement/를 참조한다. 웹 응용을 신규로 개발할 경우에는 본 절의 내용을 참고하여 통합을 진행한다.

코드 1-12	WEB-INF/securityContext.xml

```
<security:http security="none" pattern="/favicon.ico" />
<security:http security="none" pattern="/images/**" />
<security:http security="none" pattern="/css/**" />
<security:http security="none" pattern="/logout.jsp" />
```

- 코드 1-12의 security="none"은 SAML 인증이 필요하지 않은 PATH를 지정할 때 사용한다. 이미지 경로, CSS 경로 등 SAML 인증을 받지 않고 접근해야 하는 PATH를 지정한다.

코드 1-13	WEB-INF/securityContext.xml

```
<security:http pattern="/saml/web/**" use-expressions="false">
  <security:access-denied-handler error-page="/saml/web/metadata/login" />
  <security:form-login login-processing-url="/saml/web/login" login-page="/saml/web/metadata/login"
default-target-url="/saml/web/metadata" />
  <security:intercept-url pattern="/saml/web/metadata/login" access="IS_AUTHENTICATED_ANONYMOUSLY" />
  <security:intercept-url pattern="/saml/web/**" access="ROLE_ADMIN" />
  <security:custom-filter before="FIRST" ref="metadataGeneratorFilter" />
</security:http>

<security:http entry-point-ref="samlEntryPoint" use-expressions="false">
  <security:intercept-url pattern="/saml/web//**" access="IS_AUTHENTICATED_FULLY" />
  <security:custom-filter before="FIRST" ref="metadataGeneratorFilter" />
  <security:custom-filter after="BASIC_AUTH_FILTER" ref="samlFilter" />
</security:http>

<bean id="samlFilter" class="org.springframework.security.web.FilterChainProxy">
  <security:filter-chain-map request-matcher="ant">
    <security:filter-chain pattern="/saml/login/**" filters="samlEntryPoint" />
    <security:filter-chain pattern="/saml/logout/**" filters="samlLogoutFilter" />
    <security:filter-chain pattern="/saml/metadata/**" filters="metadataDisplayFilter" />
    <security:filter-chain pattern="/saml/SSO/**" filters="samlWebSSOProcessingFilter" />
    <security:filter-chain pattern="/saml/SSOHoK/**" filters="samlWebSSOHoKProcessingFilter" />
    <security:filter-chain pattern="/saml/SingleLogout/**" filters="samlLogoutProcessingFilter" />
    <security:filter-chain pattern="/saml/discovery/**" filters="samlIDPDiscovery" />
  </security:filter-chain-map>
</bean>
```

- 사용자 인증을 받아야 접근할 수 있는 PATH를 정의하고 해당 PATH에 접근했을 때 어떻게 처리해야 되는지 필터를 구현/설정해야 한다(코드 1-13 참조).

코드 1-14	WEB-INF/securityContext.xml

```
<bean id="successRedirectHandler"
class="org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler">
   <property name="defaultTargetUrl" value="/samlhandler.jsp" />
</bean>

<bean id="failureRedirectHandler"
class="org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler">
   <property name="useForward" value="true" />
   <property name="defaultFailureUrl" value="/error.jsp" />
</bean>

<bean id="successLogoutHandler"
class="org.springframework.security.web.authentication.logout.SimpleUrlLogoutSuccessHandler">
   <property name="defaultTargetUrl" value="/logout.jsp" />
</bean>
```

- 코드 1-14는 로그인과 로그아웃 요청의 결과에 따라 이동할 경로를 설정하는 필터이다.
- 웹 응용이 연합인증과 함께 로컬 로그인이나 OIDC/OAuth2 기반의 인증방식을 사용하는 경우에는, bean id 값(예, successRedirectHandler)과 defaultTargetUrl 값(예, /samlhandler.jsp)이 연합인증에서 설정한 값과 달라야 한다. 또한 defaultTargetUrl 값에 설정된 파일에서 사용자 인증 and/or 인가방식을 구현해야 한다.
- <참조> relayState를 통해 전달받은 endpoint URL로 사용자를 리디렉트하고자 하는 경우, 코드 1-14의 “successRedirectHandler” 부분을 다음과 같이 수정한다

```
<bean id="successRedirectHandler" class="org.springframework.security.saml.SAMLRelayStateSuccessHandler">
   <property name="defaultTargetUrl" value="/samlhandler.jsp" />
</bean>
```

※ bean id 값과 defaultTargetUrl 값을 기존 인증방식과 중복되지 않게 설정했으나 오류가 발생한다면 securityContext.xml 설정(코드 1-13 참조)에서 /saml/SSO/ 경로에 대한 필터(samlWebSSOProcessingFilter)를 제거 한 후, SAML Response 메시지를 처리할 수 있는 필터를 직접 구현해야 한다(아래 테이블을 참조). [확인 필요]기존 인증방식이 samlWebSSOProcessingFilter에서 사용하고 있는 handler 등을 동일하게 사용하고 있을 가능성이 있다.

SAML 로그인 처리를 위한 경로	http://your-service/saml/SSO
POST 매개변수명	SAMLResponse
파싱 및 검증 관련 참고 코드	https://github.com/oaeproject/SAMLParser/blob/master/src/main/java/org/sakaiproject/SAMLParser/SAMLParser.java

사용자가 로그인이 성공하면 식별정보제공자는 SAML Response 메시지(사용자 속성정보가 포함)를 HTTP-POST 방식으로 /saml/SSO 엔드포인트에 전달한다. /saml/SSO 경로에 응답메시지가 도착했을 때 해당 메시지를 처리 할 수 있는 코드를 구현해야 한다. 구현하는 코드에는 SAML Response에 대한 해석(parsing) 및 전자서명의 검증 등의 기능이 포함되어야 한다.

1.4.3. 사용자 속성 정보의 활용

웹 응용이 기존에 사용하는 로그인 방식이 있다면 SAML Response로 전달받은 사용자 속성을 기존 로그인 방식대로 HTTP 세션에 저장할 수 있다. 속성정보를 HTTP 세션에 저장하는 방법은 아래 코드를 참조한다.

코드 1-15	속성 정보 활용 코드, samlhandler.jsp 파일 참조

```
<%@page import="java.util.ArrayList"%>
<%@page import="java.util.Collections"%>
<%@page import="java.util.Collection"%>
<%@page import="org.springframework.security.saml.SAMLCredential"%>
<%@page import="org.springframework.security.core.context.SecurityContextHolder"%>
<%@page import="org.springframework.security.core.Authentication"%>
<%@page import="org.opensaml.saml2.core.Attribute"%>
<%@page import="org.springframework.security.saml.util.SAMLUtil"%>
<%@page import="org.opensaml.xml.util.XMLHelper"%>
<%
String isLogin = "";

try {
  isLogin = session.getAttribute("isLogin").toString();
} catch (Exception e) {
}

if (!isLogin.equals("true")) {
  try {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    SAMLCredential credential = (SAMLCredential) authentication.getCredentials();
    for (Attribute attribute : credential.getAttributes()) {
      String name = attribute.getName();
      String values = credential.getAttributeAsString(name);
      session.setAttribute(name, values);
    }
    session.setAttribute("isLogin", "true");
    isLogin = session.getAttribute("isLogin").toString();
  } catch (Exception e) {
  }
}
%>
```

1.4.4. SAML Binding 주소 설정

중요

HTTPS의 사용 등

· 바인딩된 Endpoint의 주소는 반드시 https가 적용되어야 함

· https(443)을 사용할 경우, 포트번호가 바인딩된 Endpoint 주소에 나타나지 않도록 설정

Spring Security SAML은 WAS에 설정된 바인딩 주소(eg. http://127.0.0.1:8080)를 참조하여 SAML Endpoint를 설정한다. WS를 통해 WAS에 접근할 경우, 바인딩된 Endpoint 주소(eg. http://127.0.0.1:8080)와 WS의 주소(https://your-domain.com)가 다르기 때문에 SAML Endpoint mismatch 오류가 발생한다. 오류를 해결하기 위해서는 Spring Security SAML의 SAML Endpoint를 WAS의 주소가 아닌 WS의 주소로 설정해야 한다. 설정을 적용하는 방법은 아래 코드를 참조한다.

코드 1-16	WEB-INF/securityContext.xml

```
<bean id="metadataGeneratorFilter"
class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
  <constructor-arg>
    <bean class="org.springframework.security.saml.metadata.MetadataGenerator">
      <property name="entityBaseURL" value="https://your-domain.com" />
      <property name="entityId" value="https://your-domain.com/sp/spring" />
      <property name="extendedMetadata">
        <bean class="org.springframework.security.saml.metadata.ExtendedMetadata">
          <property name="signMetadata" value="false" />
          <property name="idpDiscoveryEnabled" value="true" />
        </bean>
      </property>
    </bean>
  </constructor-arg>
</bean>
```

2. Shibboleth for Java Web Applications

본 장에서는 웹 서버와 웹 응용서버가 분리되어 있고 Apache HTTPD가 웹 서버로 활용되는 CentOS 7(Apache 2.4) 환경에서 Shibboleth를 이용해 연합인증을 구현하는 방법을 설명한다. Shibboleth는 Apache, IIS, FastCGI에서 동작한다.

※ CentOS를 이용할 경우, selinux를 비활성하거나 Shibboleth가 예외처리될 수 있도록 설정해야 한다.

```
$ getenforce
$ setenforce 0
$ nano /etc/sysconfig/selinux
  SELINUX=permissive
```

2.1 yum을 이용한 Shibboleth 설치 및 환경설정

이하, Apache 2.4가 설치되어 있다는 가정 하에 설명한다.

Shibboleth를 다음과 같이 설치한다.

```
$ yum install ca-certificates openssl

// shibboleth repository를 생성
$ nano /etc/yum.repos.d/shibboleth.repo
[shibboleth]
name=Shibboleth (CentOS_7)
# Please report any problems to https://issues.shibboleth.net
type=rpm-md
mirrorlist=https://shibboleth.net/cgi-bin/mirrorlist.cgi/CentOS_7
gpgcheck=1
gpgkey=https://download.opensuse.org/repositories/security:/shibboleth/CentOS_7/repodata/repomd.xml.key
enabled=1

$ yum update -y
// 64비트 OS일 경우
$yum install shibboleth.x86_64 -y
```

2.1.1 Apache 설정

다음 예를 참조하여 (구축된 서버의 IP 주소나 도메인명에 맞게) 호스트를 설정한다.

```
$ nano /etc/hosts
 192.168.xx.yy sp.example.org sp
```

방화벽은 항상 443포트를 개방해야 한다. 80번 포트는 localhost로만 개방하도록 설정하고 모든 HTTP 트래픽은 HTTPS로 리다이렉션 될 수 있도록 설정한다.

<중요> 항상 HTTPS로 통신해야 한다.

```
$ nano /etc/httpd/conf/httpd.conf
 Listen 127.0.0.1:80

$ nano /etc/httpd/conf.d/000-default.conf
<VirtualHost *:80>
    ServerName "sp.example.org"
    Redirect permanent "/" "https://sp.example.org/"
    RedirectMatch permanent ^/(.*)$ https://sp.example.org/$1
</VirtualHost>

$ service httpd restart
```

사용자가 웹 브라우저를 통해 특정 디렉토리로 접근했을 때, Apache가 Shibboleth 인증을 수행하도록 다음 예를 참조하여 Apache를 설정한다. 아래 예는 사용자가 https://sp.example.org/secure로 접근했을 때, Shibboleth 인증을 수행한다.

코드 2-1	Apache 설정
<Location /secure> AuthType shibboleth # ShibCompatWith24 On ShibRequestSetting requireSession 1 require shib-session </Location> $ service httpd restart	

2.1.2 Shibboleth 설정

다음 예를 참조하여 (구축된 서버의 IP 주소나 도메인명에 맞게) 호스트를 설정한다.

KAFE에서는 서비스제공자의 개체식별자(entityId)에 대해서 다음과 같은 표기법을 권장하고 있다.

https://[FQDN]/sp/[SAML software] (예, https://sp.example.org/sp/shibboleth)

코드 2-2	Shibboleth2.xml의 설정

```
$ nano /etc/shibboleth/shibboleth2.xml
...
<ApplicationDefaults entityID="https://sp.example.org/sp/shibboleth"
      REMOTE_USER="eppn persistent-id targeted-id" attributePrefix="AJP_">

cipherSuites="DEFAULT:!EXP:!LOW:!aNULL:!eNULL:!DES:!IDEA:!SEED:!RC4:!3DES:!kRSA:!SSLv2:!SSLv3:
!TLSv1:!TLSv1.1">
...
<Sessions lifetime="28800" timeout="3600" relayState="ss:mem"
            checkAddress="false" handlerSSL="true" cookieProps="https">
...
<CredentialResolver type="File" key="shibsp.key" certificate="shibsp.crt"/>
...
<SSO discoveryProtocol="SAMLDS" discoveryURL="https://ds.kreonet.net/kafe">
   SAML2 SAML1
</SSO>
...
<Errors supportContact="support@example.org"
         helpLocation="/about.html"
         styleSheet="/shibboleth-sp/main.css"/>
...
<MetadataProvider                                                    type="XML"
url="https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-testfed/metadata.xml"
                        legacyOrgName="true"              backingFilePath="kafe-test-metadata.xml"
maxRefreshDelay="7200">
       <MetadataFilter type="Signature" certificate="kafe-cert.crt"/>
       <MetadataFilter type="RequireValidUntil" maxValidityInterval="864000" />
</MetadataProvider>

<AttributeExtractor type="XML" validate="true" reloadChanges="true" path="attribute-map.xml"/>
```

- AJP 규약을 사용할 경우, attributePrefix="AJP_"를 추가한다.
- kafe-cert.crt 파일을 다음 주소로부터 내려 받아 /etc/shibboleth에 저장한다.

  https://fedinfo.kreonet.net/cert/kafe-fed.crt
- shibsp.key와 shibsp.crt의 생성

```
$ cd /etc/shibboleth
$ openssl req -newkey rsa:4096 -new -x509 -days 3650 -nodes -text -out shibsp.crt -keyout shibsp.key
$ chmod 644 shibsp.key shibsp.crt
```

• KAFE에서 제공하는 중앙형 탐색서비스를 이용할 경우, 탐색서비스 URL주소를 discoveryURL="https://ds.kreonet.net/kafe"와 같이 설정한다.

• KAFE에서 제공하는 연합 메타데이터(test federation)를 이용할 경우, MetadataProvider를 다음과 같이 설정한다.

url="https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-testfed/metadata.xml"

※ 시험용 식별정보제공자(https://testidp.kreonet.net)를 연동할 경우, shibboleth2.xml 파일의 MetadataProvider를 다음과 같이 설정한다.

```
<MetadataProvider type="XML" validate="false"
uri="https://testidp.kreonet.net/simplesaml/saml2/idp/metadata.php"
backingFilePath="testidp-metadata.xml" reloadInterval="7200">
</MetadataProvider>
```

※ 서비스제공자가 필요로 하는 사용자 속성정보를 추가할 경우, shibboleth2.xml 파일에 요구속성을 다음 예를 참조하여 설정한다.

```
<Handler type="MetadataGenerator" Location="/Metadata" signing="false">
<md:AttributeConsumingService index="1">
<md:ServiceName xml:lang="en">Liferay Test</md:ServiceName>
<md:RequestedAttribute FriendlyName="displayName"
Name="urn:oid:2.16.840.1.113730.3.1.241"/>
<md:RequestedAttribute FriendlyName="mail"
Name="urn:oid:0.9.2342.19200300.100.1.3"/>
<md:RequestedAttribute FriendlyName="uid"
Name="urn:oid:0.9.2342.19200300.100.1.1"/>
<md:RequestedAttribute FriendlyName="orgName" Name="urn:oid:2.5.4.10"/>
<md:RequestedAttribute FriendlyName="eppn"
Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6"/>
<md:RequestedAttribute FriendlyName="persistent-id"
Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10"/>
<md:RequestedAttribute FriendlyName="schacHomeOrganization"
Name="urn:oid:1.3.6.1.4.1.25178.1.2.9"/>
<md:RequestedAttribute FriendlyName="unscoped-affiliation"
```

```
Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1"/>
</md:AttributeConsumingService></md:SPSSODescriptor>
</Handler>
```

※ 새로운 사용자 속성을 추가할 경우, 다음 예를 참조하여 attribute-map.xml 파일을 수정한다.

```
$ nano /etc/shibboleth/attribute-map.xml
<!-- SCHAC attributes, uncomment to use... -->
// 아래 라인 활성화
<Attribute name="urn:oid:1.3.6.1.4.1.25178.1.2.9" id="schacHomeOrganization"/>
...
// 아래 모든 라인 활성화
<Attribute name="urn:oid:2.5.4.3" id="cn"/>
<Attribute name="urn:oid:2.5.4.4" id="sn"/>
<Attribute name="urn:oid:2.5.4.42" id="givenName"/>
<Attribute name="urn:oid:2.16.840.1.113730.3.1.241" id="displayName"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="uid"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="mail"/>
...
// 아래 라인 활성화
<Attribute name="urn:oid:2.5.4.10" id="o"/>
```

※ Shibboleth는 사용자 속성에서 Scope를 검증한다. 예를 들어, 식별정보제공자가 Scope="school.ac.kr"로 설정했다면 사용자 속성 값에 포함된 Scope이 school.ac.kr을 포함하는지 검사한다(예, 전달받은 eppn(id@Scope 형식)의 Scope 값이 myschool.ac.kr이라면 식별정보제공자가 정의한 Scope(school.ac.kr)과 다르기 때문에 오류가 발생). Scope 검증정책을 변경하기 위해서는 attribute-policy.xml 파일(아래 표 참조)을 수정한다. 아래 예에서 주석(<!-- -->)을 제거하면 schacHomeOrganization의 Scope를 검사한다.

```
$ nano /etc/shibboleth/attribute-policy.xml


<!-- Enforce that the values of schacHomeOrganization are a valid Scope. -->
<!--
<afp:AttributeRule attributeID="schacHomeOrganization">
<afp:PermitValueRule xsi:type="saml:AttributeValueMatchesShibMDScope" />
</afp:AttributeRule>
-->
```

2.1.3 Shibboleth 설정 확인

다음 명령과 URL 주소를 참조하여 Shibboleth의 상태를 관리한다.

명령어	설명
service shibd start	Shibboleth 실행
service shibd stop	Shibboleth 중지
shibd -t	Shibboleth 구동 상태

URL 주소	설명
https://[FQDN]/Shibboleth.sso/Metadata	서비스제공자의 메타데이터 확인
https://[FQDN]/Shibboleth.sso/Session	세션 정보의 확인

2.2 AJP 설정

웹 서버(Apache)에 설치된 Shibboleth는 mod_ajp 또는 mod_jk를 이용해 웹 응용서버(Tomcat 또는 JEUS)에게 속성정보를 전달한다.

※ Servlet 컨테이너 중, Tomcat은 AJP 1.3을 기본으로 지원하고 JEUS는 mod_jk만 지원하며 Jetty는 mod_ajp를 지원하지 않는다. 필요 시, Servlet 컨테이너가 AJP 또는 JK를 지원할 수 있도록 설정한다.

※ 사용자가 Servlet 컨테이너에 직접 접근할 수 없도록 Apache 환경을 설정해야 한다. 예를 들어, Tomcat 8080 포트에 대한 Apache 설정이 없으면 Shibboleth가 bypass된다.

먼저 코드 2-1의 Apache 설정을 적용하고 코드 2-2의 attributePrefix를 attributePrefix="AJP_" 로 설정한다. [중요] mod_proxy_ajp를 이용해 변수들을 전달하기 위해서는 반드시 AJP_ 프릭픽스를 이용해야 한다. 본 예에서 Shibboleth로 보호받는 디렉토리는 /secure로 설정된 상태이다. Apache에 ShibUseHeaders On을 설정하면 HTTP request headers를 이용해 변수들을 전달할 수 있지만 AJP를 이용하는 것보다 덜 안전한 것으로 알려져 있다. HTTP 헤더를 통해 전달하기 위해서는 프리픽스가 HTTP_로 변경되어야 한다.

2.2.1 AJP 패킷 사이즈

서비스제공자에게 여러개의 사용자 속성정보가 전달된다면 기본 AJP 패킷 크기인 8Kb를 초과할 수 있다. Apache와 Servlet 컨테이너 모두에서 AJP 패킷 크기를 65Kb(최대 크기임)로 증가시킨다. Tomcat의 AJP <Connector>에서 packetSize="65536"로 설정하고 Apache에서 ProxyIOBufferSize 65536으로 설정한다.

```
ProxyIOBufferSize 65536
```

2.2.2 Apache 설정

코드 2-1을 적용함으로써 Protected Resource를 설정해야 한다. 또한 Apache로 수신되는 HTTP 요청을 AJP 1.3을 통해 웹 응용서버에 전달할 수 있도록 Apache에 ProxyPass를 설정해야 한다. 다

음 예를 참조한다.

```
ProxyPass /secure ajp://localhost:8009/secure
```

3. OIDC Client for Flask Web Framework

3.1 검증 환경

본 문서의 내용은 다음 표와 같은 환경에서 검증되었다.

	상세
검증 환경	Ubuntu 18.04.4 LTS
	Python 3.6.x
	Flask 1.1.x

이용된 예제 코드 및 라이브러리는 아래 주소에서 내려 받을 수 있다. git으로 내려 받은 후 예제 코드에 사용된 라이브러리를 설치한다.

	상세
Flask 예제 저장소	https://git.kreonet.net/kafe-support/flask-oidc-client-sample

git 프로젝트에는 app.py와 requirements.txt 파일이 포함되어 있다. git 프로젝트에 포함된 requirements.txt는 app.py(예제 코드)에 필요한 라이브러리가 정의되어 있다.

```
$ pip install -r requirements.txt
```

3.2 OIDC Client 서비스 등록

OIDC Provider에서 정보를 받아올 서비스 OIDC client를 KAFE Trust Zone에 등록해야 한다. KAFE Trust Zone(https://trust.kafe.or.kr/)에서 OIDC Client 정보를 등록한다.

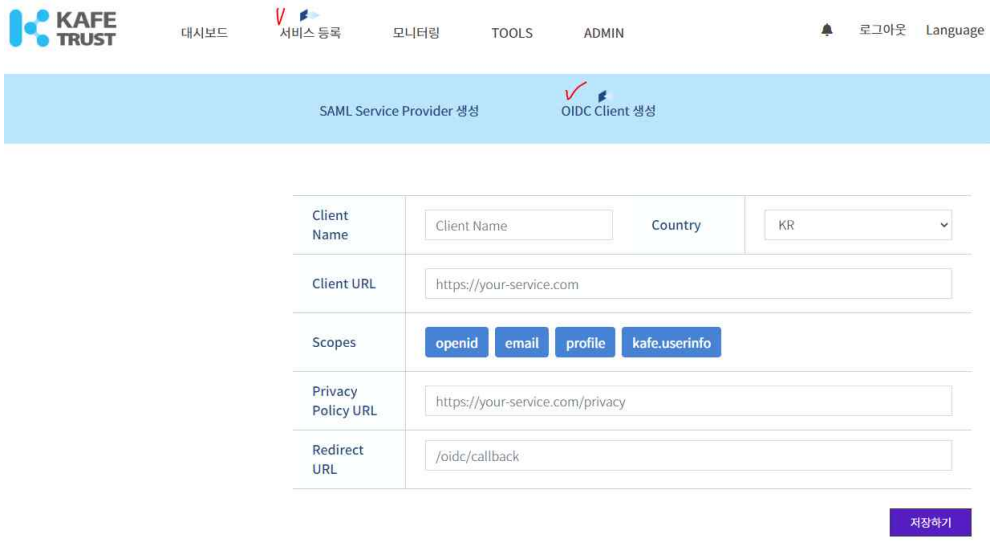
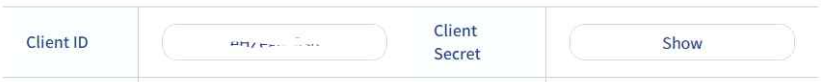


그림. OIDC client의 등록

Trust Zone에 로그인이 불가능할 경우에는 다음 정보를 support@kafe.or.kr에 전달하고 서비스 등록을 요청한다.

항목	구분	설명
Client name	필수	예, TEST_SERVICE
Client URL	필수	예, https://127.0.0.1
Scope	필수	필요한 사용자 속성값을 support@kafe.or.kr로 전달
Privacy Policy URL	프로덕션 서비스 시 필수	예, https://127.0.0.1/privacy
Redirect URL	필수	로그인 성공 후 callback 주소

OIDC Client가 등록되면 Trust Zone은 다음 그림과 같이 Client_ID와 Client_Secret을 발급한다.



3.3 자가서명 인증서 발급 및 HTTPS 설정

3.3.1 자가서명 인증서의 생성

Private KEY 생성

(command : openssl genrsa -out [파일명] 2048)

```
$ openssl genrsa -out private.key 2048
```

Public KEY 생성

(command : openssl rsa -in [private.key 파일명] -pubout -out [public.key 파일명]

```
$ openssl rsa -in private.key -pubout -out public.key
```

CSR 생성 (Certificate Signing Request - 인증서 서명 요청)

중요

Common Name

· OIDC client 서버의 도메인 이름이나 IP 주소로 설정해야 함

※ 프로덕션 서비스 시, 반드시 도메인 이름으로 설정

구분	작성 예
Country Name (국가코드)	KR
State or Province Name (시 / 도의 전체이름)	Seoul
Locality Name (시/군/구 등의 이름)	Songpa-gu
Organization (회사이름)	XXXX
Organization Unit (부서명)	Server
Common Name (SSL 인증서를 설치할 서버의 Full Domain)	www.XXXX.com

(command : openssl req -new -key [private.key 파일명] -out [.csr 파일명])

코드 3-1	CSR생성 명령어
$ openssl req -new -key private.key -out private.csr	

CSR 생성 후 인증서 발급을 위한 필요한 정보 작성

```
Country Name (2 letter code) [AU]:KR
    State or Province Name (full name) [Some-State]:Seoul
    Locality Name (eg. city) []:Seoul
    Organization Name (eg. company) [Internet Widgits Pty Ltd]:Local
    Organizational Unit Name (eg. section) []:local
    Common Name (eg. YOUR name) []:local
    Email Address []:test@test.com

Please enter the following 'extra' attributes
    To be sent with your certificate request
    A challenge password []:test
    An optional company name[]:test
```

CRT 인증서 만들기

(command : openssl req -x509 -sha256 -days [기간] -key [private key 파일명] -in [csr 파일명] -out [파일명] -days [기간])

코드 3-2	proxy 설정하기 위해 필요한 apache2 모듈 명령어
$ openssl req -x509 -days 3650 -sha256 -key private.key -in private.csr -out mycommoncrt.crt	

CRT 파일을 PEM 파일로 변환

(command : openssl x509 -inform PEM -in CRT파일명.crt > PEM파일명.pem)

코드 3-3	proxy 설정하기 위해 필요한 apache2 모듈 명령어

```
$ openssl x509 -inform PEM -in mycommoncrt.crt > mycommonpem.pem
```

생성된 mycommonpem.pem 파일은 app.py에서 사용한다. 세부 내용은 3.4를 참조한다.

3.4 OIDC Client 예제 코드

3.4.1 환경설정

3.3.1에서 생성한 pem 파일을 app.py에 등록한다.

코드 3-4	app.py 파일에 자가서명 인증서 등록

```
112 if __name__ == "__main__":
113
114         ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS)
115         ssl_context.load_cert_chain(certfile='cert/mycommonpem.pem',
keyfile='cert/private.key', password='password')
116         app.run(host='0.0.0.0', port=443, ssl_context=ssl_context)
```

CLIENT_ID와 CLIENT_SECRET을 app.py에 등록한다. CLIENT_ID와 CLIENT_SECRET는 각각 3.2 에서 생성한 Client_ID와 Client_Secret이다. app.secret_key는 세션과 쿠키관리를 위해 필요하며 값(SECRET)은 임의로 설정할 수 있다.

코드 3-5	app.py

```
import json
import os
import ssl
# Third-party libraries
from flask import Flask, redirect, request, url_for, session

from oauthlib.oauth2 import WebApplicationClient
import requests
from urllib.parse import urlparse
from urllib.parse import (
    quote as _quote, unquote as _unquote, urlencode as _urlencode,
)
CLIENT_ID = "your_client_id"
CLIENT_SECRET = "your_client_secret"
DISCOVERY_URL = (
    "https://oidc.kafe.or.kr/.well-known/openid-configuration"
)
app = Flask(__name__)
app.secret_key = "SECRET"
client = WebApplicationClient(CLIENT_ID)
```

3.4.2 사용자 속성정보의 획득

코드 3-6	app.py

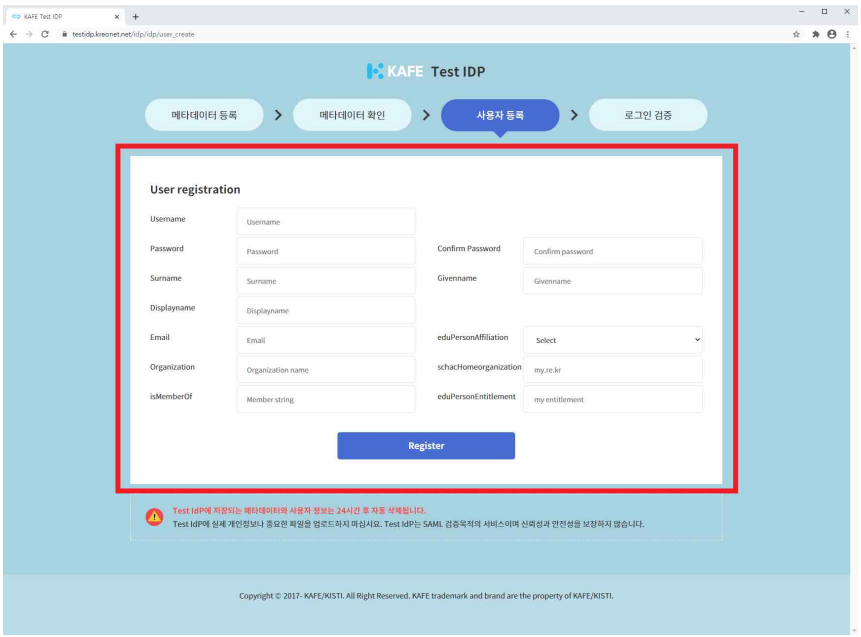
```
request_uri = client.prepare_request_uri(
        authorization_endpoint,
        redirect_uri=request.base_url + "/callback",
        scope=["openid", "email", "profile"],
)

unique_id = userinfo_response.json().get("nickname")
users_name = userinfo_response.json().get("given_name")
users_email = userinfo_response.json().get("email")
```

인증에 성공한 사용자의 속성정보(Scopes/Claims)는 위와 같은 방법으로 확보한다. 3.2의 「그림. OIDC client의 등록」에서 정의한 Scope와 코드 3-6에서 정의한 scope가 일치해야 한다. “openid, ‘email’, ‘profile’, ‘kafe.userinfo’를 모두 등록해야 필수제공 속성을 획득할 수 있다.

Scope	Claims	필수제공 속성
openid	sub	○
profile	name	○
	given_name	
	family_name	
	nickname	○
kafe.userinfo	eppn	○
	epsa	
	epen	
	ismemberof	
email	email	○
	email_verified	
address	address	
phone	phone_number	
	phone_number_verified	
offline_address	[]	

이용 가능한 속성정보(Claims)는 위 표와 같다. sub는 특정 응용서비스에서만 유효한 사용자 고유식별자이고 eppn은 모든 응용서비스에 동일하게 사용되는 고유식별자이다. sub나 eppn을 이용해 사용자를 구분할 수 있지만 eppn의 이용이 권장된다.

3.5 시험용 식별정보제공자에 사용자 등록



참고

eppn claims의 제공

시험용 식별정보제공자는 eppn을 제공하지 않는다.

시험용 식별정보제공자의 '사용자 등록' 메뉴를 클릭하면 위 그림과 같이 화면이 나타난다. 해당 식별정보제공자에서 로그인이 가능한 임시 사용자를 생성할 수 있다. 'Username' 등은 임의로 입력해도 되지만 실제 이름 등 개인정보를 입력하지 않도록 주의한다. 'Username'은 사용자 ID를 의미한다. 등록된 사용자계정은 24시간 동안만 유효하다.

	서버 주소
시험용 식별정보제공자	https://testidp.kreonet.net

3.5.1. OIDC Provider와 서비스제공자의 연동 확인

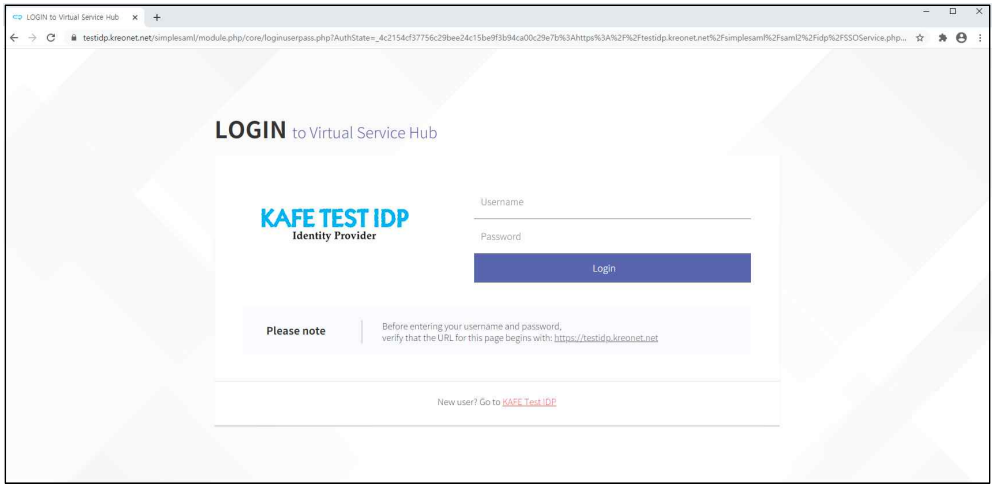
예제 프로젝트를 빌드한 후, 웹 브라우저를 이용해 예제 응용서비스에 접근한다. 아래 your-service(적색 표시)은 서비스제공자의 구축환경에 맞춰 적절히 수정한다.

(Ubuntu에서 빌드 시 HTTPS의 기본포트가 443으로 1024 미만의 포트는 root계정으로만 열 수 있으므로 실행은 root계정으로 한다.)

예제 프로그램의 접속 주소	https://your-service (local인 경우 https://127.0.0.1)

예제 프로그램 접속 후 'Login'을 클릭하면 그림 <로그인 화면>처럼 시험용 서비스 제공자의 로그

인 화면이 나타난다.



<로그인 화면>

3.5절에서 생성한 사용자 ID(Username)과 비밀번호를 입력하고 로그인을 시도한다.

로그인에 성공하면, 시험용 식별정보제공자가 전달한 사용자 속성정보를 서비스제공자에서 확인할 수 있다.

3.6 Flask Apache(using mod_wsgi) 연동

Apache 웹 서버와 웹 응용서버가 동일한 머신에 존재할 경우에는 본 절의 내용을 참조한다. WSGI를 이용해 Apache 웹 서버에 연동하기 위해 mod-wsgi 패키지 설치를 한다.

사용자가 사용하고 있는 python의 버전을 확인 후 버전에 맞게 설치해야 한다.

코드 3-8	mod-wsgi 패키지 설치 명령어

```
$sudo apt-get install libapache2-mod-wsgi-py3 (python3)
$sudo apt-get install libapache2-mod-wsgi (python2)
```

Apache와 연동하기 위한 .wsgi 파일 생성

코드 3-9	application.wsgi

```
import sys
sys.path.insert(0, '/path/to/the/yourapplicationDirectory')
from yourapplication import app as application
```

Apache 설정

코드 3-10	Apache 설정 (000- default.conf)

```
$vi /etc/apache2/sites-available/000-default.conf


<VirtualHost *:443>
    SSLEngine On
    SSLCertificateFile /path/to/the/pemfile.pem
    SSLCertificateKeyFile /path/to/the/keyfile.key

    WSGIDaemonProcess yourapplicationDirectory
    WSGIApplicationGroup yourapplicationDirectory
    WSGIScriptAlias / /path/to/the/wsgiFile.wsgi

    <Directory /path/to/the/yourapplicationDirectory>
        Order deny,allow
        Allow from all
        Require all granted
    </Directory>
</VirtualHost>
```

4. OIDC Client for Spring framework

4.1 검증 환경

본 문서의 내용은 다음 표와 같은 환경에서 검증되었다.

	상세
검증 환경	JDK 14.0.2
	Tomcat 9.0 버전
	Maven

이용된 예제 코드는 아래 주소에서 내려 받을 수 있다. git 또는 zip 파일을 내려 받은 후 IDE(Integrated Development Environment)에서 import 기능을 사용하여 프로젝트를 불러온다.

	상세
Spring 예제 저장소	https://git.kreonet.net/kafe-support

4.2 OIDC Client의 등록 및 설정

본 장의 3.2절을 참조해 OIDC client를 등록하고 생성한 OIDC client의 정보를 HomeController.java에 설정한다. authorization_endpoint와 token_endpoint 및 userinfo_endpoint 정보는 https://oidc.kafe.or.kr/.well-known/openid-configuration를 참조한다.

코드 4-1 oidcClient/HomeController.java

```
@Controller
public class HomeController{
    private static String client_id = client_id;
    private static String client_secret = client_secret;
    private static String callback_url = callbak_url;
    private static String authorizationendpoint = authorization_endpoint;
    private static String tokenendpoint = token_endpoint;
    private static String userinfoendpoint = userinfo_endpoint;
```

4.3 OIDC Provider와 서비스제공자의 연동 확인

본 장의 3.5에서 설명한대로 시험용 사용자를 등록하고 예제 프로젝트를 빌드한 후, 웹 브라우저를 이용해 예제 응용서비스에 접근한다. 붉게 표시된 부분은 서비스제공자의 구축환경에 맞춰 적절히 수정한다.

예제 프로그램의 접속 주소	http://localhost:8080

Chapter 3. 계정연합 참여

1. 계정연합(KAFE) 연동

https://testidp.kreonet.net에서 기능검증이 끝난 후, KAFE(support@kafe.or.kr)에 제출한 서비스제공자의 메타데이터는 회원기관에 자동으로 배포(최대 1일 소요)된다. 서비스제공자의 메타데이터는 다음 사항을 충족해야 한다.

- 메타데이터에는 IP 주소가 포함되지 않아야 한다.
- 메타데이터에 포함된 모든 URL 주소는 https로 시작되어야 하며 실제로 접근 가능해야 한다.
- 메타데이터에 포함된 인증서는 SHA256 이상의 서명알고리즘과 2048 이상의 키 길이를 가져야 한다.

Tip

KAFE 연합 메타데이터

KAFE 연합 메타데이터는 다수의 식별정보제공자와 서비스제공자의 개별 메타데이터를 포함하고 있다. 연합 메타데이터는 검증(Test), 프로덕션(Production), 에듀게인(EduGAIN)으로 구분된다. 서비스제공자가 에듀게인에 참여하고 에듀게인 용 메타데이터를 등록하면 국외 사용자가 국외 식별정보제공자를 통해 해당 서비스제공자에 로그인할 수 있다.

서비스제공자는, 개별 메타데이터를 KAFE에 제출한 후, KAFE 계정연합에서 배포 중인 연합 메타데이터(Federation metadata)를 서버에 등록해야 한다. KAFE에서 제공하는 연합 메타데이터는 'test federation'용과 'production federation'용으로 구분된다. 모든 서비스제공자는 'test federation' 용 메타데이터를 이용해 응용서비스가 정상적으로 동작하는지 확인해야 한다(즉, 'test federation'에 연동된 식별정보제공자를 통해 정상적으로 로그인할 수 있고 식별정보제공자로부터 속성정보를 획득할 수 있는지 확인 필요). 서비스제공자가 정회원에 가입하면 'production federation' 용 연합 메타데이터를 이용할 수 있다. 참여방법은 https://www.kafe.or.kr/join을 참조한다.

연합 메타데이터의 배포 URL 주소는 다음 표(https://www.kafe.or.kr/join 참조)와 같다.

test federation	https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-testfed/metadata.xml
production federation	https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-profed/metadata.xml

'test federation'용 연합 메타데이터를 서비스제공자에 등록하는 방법은 다음과 같다.

1.1 Spring Security SAML

코드 1-1	WEB-INF/securityContext.xml

```
<bean id="metadata" class="org.springframework.security.saml.metadata.CachingMetadataManager">
```

```
  <constructor-arg>
    <list>
      ...
      <bean class="org.springframework.security.saml.metadata.ExtendedMetadataDelegate">
        <constructor-arg>
          <bean class="org.opensaml.saml2.metadata.provider.HTTPMetadataProvider">
            <constructor-arg>
              <value type="java.lang.String">
                https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-testfed/metadata.xml
              </value>
            </constructor-arg>
            <constructor-arg>
              <value type="int">5000</value>
            </constructor-arg>
            <property name="parserPool" ref="parserPool" />
          </bean>
        </constructor-arg>
        <constructor-arg>
          <bean class="org.springframework.security.saml.metadata.ExtendedMetadata"></bean>
        </constructor-arg>
        <property name="metadataTrustCheck" value="true" />
      </bean>
      ...
    </list>
  </constructor-arg>
</bean>
```

코드 1-1의 metadataTrustCheck이 true로 설정된 경우, KAFE에서 제공하는 인증서를 이용해 연합 메타데이터의 무결성을 검증할 수 있다. KAFE에서 제공하는 서명용 인증서는 서비스제공자의 keystore에 등록되어야 한다. 서명용 인증서는 아래 주소에서 내려받을 수 있다.

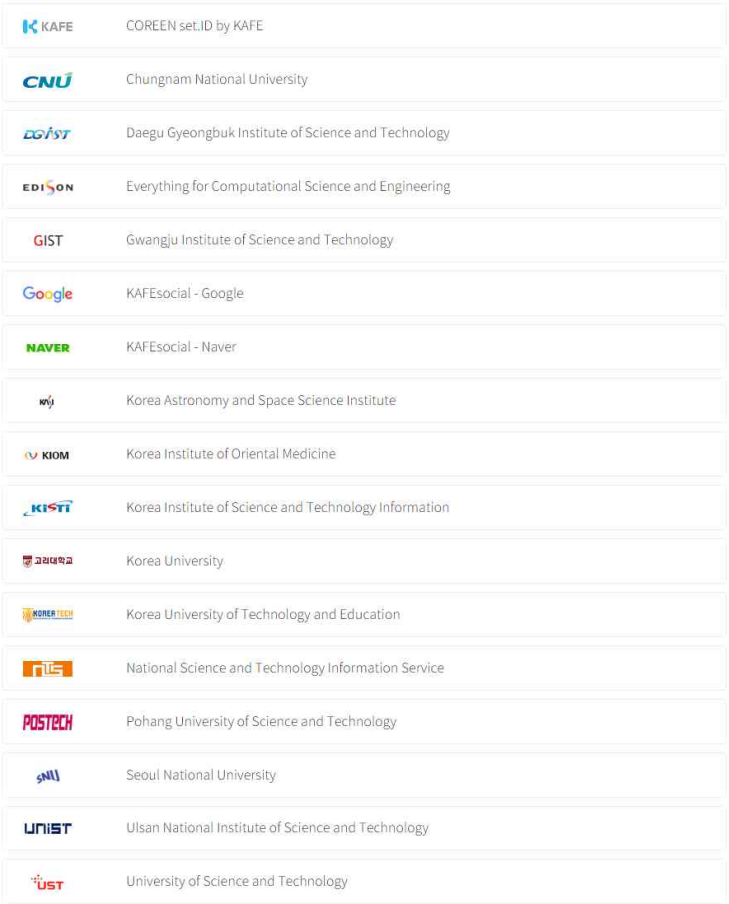
https://fedinfo.kreonet.net/cert/kafe-fed.crt

코드 1-2	keystore에 KAFE 서명용 인증서를 등록하는 방법(Linux)

$ cd src/main/resources/security/
$ wget https://fedinfo.kreonet.net/cert/kafe-fed.crt
$ keytool -importcert -keystore samlKeystore.jks -storepass [keystore 암호] -alias kafe-fed -file kafe-fed.crt	

코드 1-3	keystore에 KAFE 서명용 인증서를 등록하는 방법(Windows)
$ "C:\Program Files\Java\jdk1.8.0_131\jre\bin\keytool.exe" -importcert -keystore samlKeystore.jks -storepass nalle123 -alias kafe-fed -file kafe-fed.crt	

1.2. 탐색서비스 설정



<중앙형 식별정보제공자 탐색서비스>

서비스제공자는 로그인할 기관(식별정보제공자)을 사용자가 선택할 수 있도록 해야 한다(예, 서울대 학생은 서울대의 식별정보제공자를 통해 로그인해야 함). 서비스제공자가 하나의 식별정보제공자와 1:1로 연동될 경우에는 탐색서비스가 필요하지 않다. 하지만, 두 개 이상의 식별정보제공자와 연동하기 위해서는 탐색서비스를 이용해야 한다. 서비스제공자는 KAFE에서 제공하는 중앙형 탐색서비스(Central discovery service)를 이용하거나 자체적으로 내장형 탐색서비스(Embedded discovery service)를 구현할 수 있다. 내장형 탐색서비스는 중앙형 탐색서비스의 운용 상태에 영향을 받지 않는다는 장점이 있다.

1.2.1 Spring Security SAML

KAFE에서 제공하는 중앙형 탐색서비스를 이용하기 위해서 securityContext를 다음과 같이 설정한다.

코드 2-1	WEB-INF/securityContext.xml
<bean id="metadataGeneratorFilter"	

```
  class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
  <constructor-arg>
    <bean
      class="org.springframework.security.saml.metadata.MetadataGenerator">
      <property name="entityId" value="http://localhost:8080/sp/spring" />
      <property name="extendedMetadata">
        <bean
          class="org.springframework.security.saml.metadata.ExtendedMetadata">
          <property name="signMetadata" value="false" />
          <property name="idpDiscoveryEnabled" value="true" />
          <property name="idpDiscoveryURL" value="https://ds.kreonet.net/kafe" />
        </bean>
      </property>
    </bean>
  </constructor-arg>
</bean>
```

중요

중앙형 탐색서비스 이용 시, URL 매개변수 확인 필요

· [아래 참조] URL 매개변수로 'entityID', 'return', 'returnIDParam'이 모두 포함되는지 확인

탐색서비스가 서비스제공자와 정상적으로 연동되면 로그인 버튼 등을 클릭했을 때 위 <중앙형 식별정보제공자 탐색서비스>와 같은 화면을 나타난다.

탐색서비스가 나타나면 웹 브라우저의 주소창에서 다음 사항을 확인한다.

· URL 매개변수로 'entityID', 'return', 'returnIDParam'를 모두 포함

3개의 매개변수가 모두 포함되어 있어야 탐색서비스가 정상적으로 동작한다. 3개의 매개변수가 포함되어 있지 않다면 웹 응용서비스의 로그인 버튼이 갖는 링크를 아래와 같이 설정해 매개변수를 수동으로 포함시킨다.

로그인 호출 URL	https://ds.kreonet.net/kafe?entityID=[서비스제공자의 entityID]&return=[탐색에 성공한 후 리턴할 주소]&returnIDParam=idp
예시	https://ds.kreonet.net/kafe?entityID=https://myservice.kr/sp/spring&return=https://myservice.kr/saml/login&returnIDParam=idp