JAVA 웹 응용의 SAML 2.0 연동

일자	2016년 10월 14일
부서	슈퍼컴퓨팅본부 첨단연구망응용지원실
작성자	이경민, 곽재승

목 차

Java with Spring Security SAML extension

제 1 장 개발환경

본 매뉴얼에서 기술할 Spring Security SAML Extension 은 다음 환경에서 검증되었다.

항목	비고
OS	Windows 7 Pro 64bit
Java	JDK 1.8.0.60
Web container	Tomcat 8.0.15
IDE	Netbeans 8.1
Spring Security SAML version	1.0.2

참고 페이지는 다음과 같다.

항목	비고
Spring Security SAML extension	http://projects.spring.io/spring-security-saml/
Spring Security SAML Sample	https://github.com/spring-projects/spring-security-saml
Spring Security	http://projects.spring.io/spring-security/

제 2 장 Spring Security SAML extension

Spring Security SAML extension 은 Spring Security 의 SAML 확장 모듈이다. Spring Security 는 OAuth 등 다양한 확장 모듈을 가지고 있다. Spring Security SAML extension 을 이용하면 SAML 기반의 사용자 인증이 가능하다.

사용자 인가나 Attribute filter 등의 적용을 위해서는 Spring Security 에 포함된 클래스를 상속받아 재 작성해야 된다. SAML extension 을 이용하기 위해서는 Spring Security 에 대한 지식이 필요하다. 본 매뉴얼은 SAML extension 을 이용한 사용자 인증이 주목적이기 때문에 Spring Security 에 대해서는 다루지 않는다. Spring Security 와 관련된 내용은 제 1 장의 참고 페이지를 참조한다.
아래 그림은 사용자 인증에 필요한 주요 handler와 filter의 적용순서를 보여준다. 사용자 로그인 요청 시 /saml/login/?RelayState=/my/login/servlet?returnTo=returnToURL&param1=p1&...와

같은 스트링이 ID 제공자에게 전달된다고 가정한다.

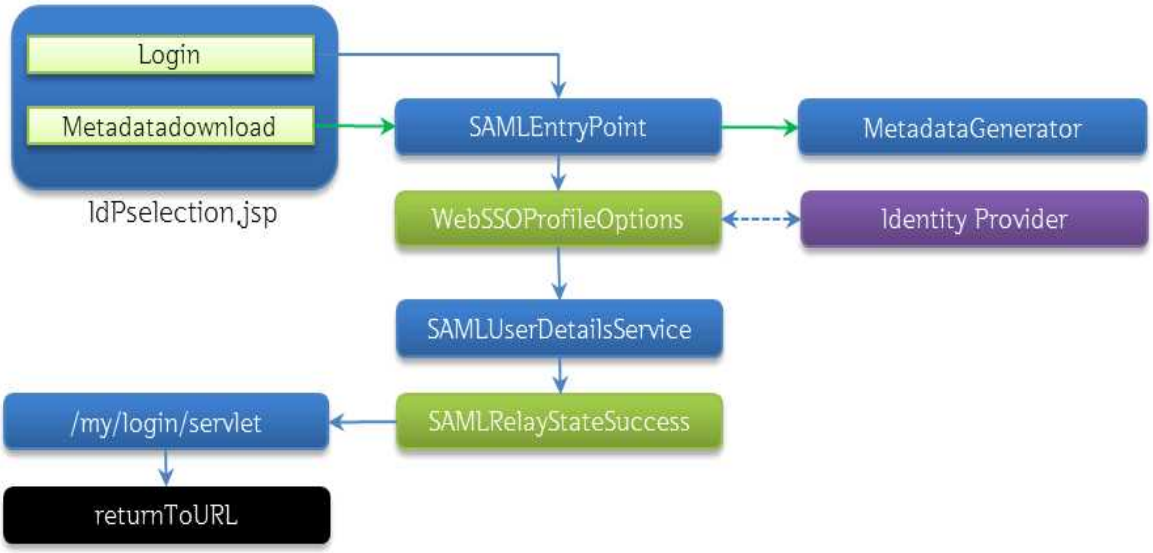


그림. 핸들러 및 필터의 적용 순서

파란색 박스는 예제 Web app의 배포를 위해 새롭게 구현한 부분이다. 파랜색 박스에 해당되는 부분을 서비스 제공자 측 개발자가 수정/구현해야 한다. 모듈별 기능은 다음과 같다.

모듈명	기능
IdPselection.jsp	ID 제공자를 선택하거나 서비스 제공자의 메타데이터를 다운로드 받음
SAMLEntryPoint	URL에 포함된 RelayState 값을 서비스 제공자의 환경에 맞게 수정
MetadataGenerator	서비스 제공자의 메타데이터에 포함되어야 할 내용을 추가
SAMLUserDetailsService	사용자 속성정보를 추출하고 Class object에 저장하여 리턴
SAMLRelayStateSuccess	RelayState를 기반으로 페이지 redirect · RelayState가 URL string에 포함된 경우: RelateState에 지정된 페이지(ex. /my/login/servlet) · RelayState가 URL string에 포함되지 않은 경우: securityContext.xml에 설정되 기본 URL로 redirect
/my/login/servlet	로그인에 성공한 사용자를 대상으로 웹 응용이 처리해야 하는 작업을 구현 (ex. returnTo를 참고한 페이지 redirect, 사용자 인가 등)

제 3 장 Spring Security SAML 의 적용

3.1 Spring Security SAML extension 설치

Spring Security SAML extension 을 설치할 Web app 의 pom.xml 파일에 다음 내용을 입력한다.

```
// Spring Security SAML 설치
<dependency>
  <groupId>org.springframework.security.extensions</groupId>
  <artifactId>spring-security-saml2-core</artifactId>
  <version>1.0.2.RELEASE</version>
  <scope>compile</scope>
</dependency>

// Spring Security 설치
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>3.1.3.RELEASE</version>
  <scope>compile</scope>
</dependency>
```

3.2 securityContext.xml 생성

WEB-INF/securityContext.xml 파일을 생성한다. securityContext.xml 은 Spring Security 와 Spring Security SAML 에 필요한 설정 정보를 갖는다.

https://github.com/spring-projects/spring-security-saml에 접속한 후 sample/src/main/webapp/WEB-INF/securityContext.xml 파일을 구현 중인 Web app 의 securityContext.xml 로 복사한다.

3.3 ID 제공자 Metadata Provider 설정

서비스 제공자는 ID 제공자의 메타데이터를 주기적으로 갱신해야 한다. ID 제공자도 서비스 제공자의 메타데이터를 주기적으로 갱신한다. Spring Security SAML extension 은 ID 제공자의 메타데이터를 주기적으로 다운로드해 갱신하기 위해 다수의 Provider 클래스를 제공한다. 본 문서에서는 3 가지 방법을 소개한다. securityContext.xml 에서 org.springframework.security.CachingMetadataManager 에서 ID 제공자의 Metadata Provider 를 설정할 수 있다. 아래 그림은 FileBackedHTTPMetadataProvider 를 설정하는 방법을 예시한다.

그림. Metadata Provider의 설정

3.3.1 FilesystemMetadataProvider

ID 제공자의 메타데이터를 파일에서 읽어와 반영한다. 특정 URL 에서 주기적으로 다운로드하지 않으며 저장된 파일을 정적으로 이용한다. 아래 글상자는 FilesystemMetadataProvider 의 예제이다.

```
// "그림. Metadata Provider의 설정"을 참조
<bean class=org.opensaml.saml2.metadata.provider.FilesystemMetadtaProvider>
  <constructor-arg>
      <value type="java.io.File"> 파일이 저장될 경로 </value>
  </constructor-arg>
  ...
</bean>
```

3.3.2 HTTPMetadataProvider

ID 제공자의 메타데이터를 웹 URL 에서 가져온다. 다음 글상자는 HTTPMetadataProvider 의 사용 예시이다.

```
// "그림. Metadata Provider의 설정"을 참조
<bean class=org.opensaml.saml2.metadata.provider.HTTPMetadataProvider>
  <constructor-arg>
      <value type="java.io.String"> metadata URL </value>
      <value type="int"> timeout 시간(밀리초) </value>
  </constructor-arg>
  ...
</bean>
```

3.3.3 FileBackedHTTPMetadataProvider

ID 제공자의 메타데이터를 웹 URL 에서 다운로드 한 후 파일에 저장한다. 이후 메타데이터 갱신 시 URL 에 접근하지 못할 경우 저장된 파일에서 ID 제공자의 메타데이터를 읽어온다. KAFE 에서 제공하는 예제 Web app 는 FileBackedHTTPMetadataProvider 를 적용했다.

```
// "그림. Metadata Provider의 설정"을 참조
<bean class=org.opensaml.saml2.metadata.provider.HTTPMetadataProvider>
  <constructor-arg>
      <value type="java.io.String"> metadata URL </value>
      <value type="int"> timeout 시간(밀리초) </value>
      <value type="java.lang.String"> 파일이 저장될 경로 </value>
  </constructor-arg>
  <property name="parserPool" ref="parserPool" />
  <property name="requireValidMetadata" value="true" />
  <property name="minRefreshDelay" value="최소 refresh 주기(밀리초)" />
  <property name="maxRefreshDelay" value="최대 refresh 주기(밀리초)" />
</bean>
```

글상자 하단의 property 는 Metadata 파일의 갱신 주기를 나타낸다. 최소 1 시간에서 최대 6 시간 사이에 갱신되도록 설정한다. KAFE 에서 제공하는 federation metadata 의 URL 주소는 https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-testfed/metadata.xml 이다.

3.4 서비스 제공자 Metadata 의 설정

서비스 제공자의 메타데이터는 ID 제공자에게 전달되어야 한다. 서비스 제공자의 메타데이터를 설정하기 위해 MetadataGenerator 클래스를 수정한다. securityContext.xml 에서 org.springframework.security.saml.metadata.MetadataGeneratorFilter 를 찾는다.

<MetadataGenerator> bean 아래에 다음 항목을 설정한다. 서비스 제공자의 entityID 는 URL/sp/spring (예. https://example.com/sp/spring)의 형식으로 기록한다. 아래 글상자의 "spring.security.saml"이 새로 만들어질 package 이름이 된다.

```
<bean                                                    id="metadataGeneratorFilter"
class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
  <constructor-arg>
      <bean class="spring.security.saml.MetadataGenerator">
        <property name="entityId" value="서비스제공자의 entityID" />
        <property name="entityBaseURL" value="웹 응용의 URL" />
        <property name="nameID" value="PERSISTENT" />
        <property name="extendedMetadata">
          <bean class="org.springframework.security.saml.metadata.ExtendedMetadata">
              <property name="idpDiscoveryEnabled" value="true" />
          </bean>
        </property>
```

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

아래 그림파일을 참조해 설정한다.

```
<!-- Filter automatically generates default SP metadata -->
<bean id="metadataGeneratorFilter" class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
    <constructor-arg>
        <bean class="spring.security.saml.MetadataGenerator">
            <property name="entityId" value="http://myspring-sp.kreonet.net/sp/spring-securtiy-saml" />
            <property name="entityBaseURL" value="http://                8084/Spring-Security-SAML" />
            <property name="nameID" value="PERSISTENT" />
            <property name="extendedMetadata">
                <bean class="org.springframework.security.saml.metadata.ExtendedMetadata">
                    <property name="idpDiscoveryEnabled" value="true" />
                </bean>
            </property>
        </bean>
    </constructor-arg>
</bean>
```

그림 MetadataGenerator bean의 설정

서비스 제공자가 필요로 하는 사용자 속성정보(email 주소 등)를 메타데이터에 등록해야 한다. 이 정보는 securityContext 에서 설정할 수 없으며 org.springframework.security.saml.metadata.MetadataGenerator 를 재 정의해서 사용해야 한다. MetadataGenerator bean 에서 패키지 이름을 spring.security.saml 로 설정했다면 다음과 같이 MetadataGenerator 클래스를 생성한다.

```
package spring.security.saml;

import java.util.Collection;
import org.opensaml.common.SAMLObjectBuilder;
import org.opensaml.saml2.metadata.AttributeConsumingService;
import org.opensaml.saml2.metadata.LocalizedString;
import org.opensaml.saml2.metadata.RequestedAttribute;
import org.opensaml.saml2.metadata.SPSSODescriptor;
import org.opensaml.saml2.metadata.ServiceName;

public class MetadataGenerator extends org.springframework.security.saml.metadata.MetadataGenerator {

    //서비스명을 영문으로 입력
    private String WebAppServiceName = "서비스명(영문)";

    @Override
    protected SPSSODescriptor buildSPSSODescriptor(String entityBaseURL, String entityAlias, boolean
requestSigned, boolean wantAssertionSigned, Collection<String> includedNameID) {
        SPSSODescriptor descriptor = super.buildSPSSODescriptor(entityBaseURL, entityAlias, requestSigned,
wantAssertionSigned, includedNameID);
        descriptor.getAttributeConsumingServices().add(createRequriredAttribute());
        //descriptor.setOrganization(createOrganization());
```

```
        return descriptor;
    }

    private AttributeConsumingService createRequriredAttribute() {
        SAMLObjectBuilder<AttributeConsumingService> builder =
(SAMLObjectBuilder<AttributeConsumingService>)
builderFactory.getBuilder(AttributeConsumingService.DEFAULT_ELEMENT_NAME);
        AttributeConsumingService service = builder.buildObject();

        SAMLObjectBuilder<ServiceName> ServiceNameBuilder = (SAMLObjectBuilder<ServiceName>)
builderFactory.getBuilder(ServiceName.DEFAULT_ELEMENT_NAME);
        ServiceName serviceName = ServiceNameBuilder.buildObject();
        serviceName.setName(new LocalizedString(this.WebAppServiceName, "en"));
        service.getNames().add(serviceName);

        SAMLObjectBuilder<RequestedAttribute> RequestAttributeBuilder =
(SAMLObjectBuilder<RequestedAttribute>)
builderFactory.getBuilder(RequestedAttribute.DEFAULT_ELEMENT_NAME);
        RequestedAttribute email = RequestAttributeBuilder.buildObject();
        email.setIsRequired(true);
        email.setFriendlyName("email");
        email.setName("urn:oid:0.9.2342.19200300.100.1.3");
        email.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(email);

        RequestedAttribute uid = RequestAttributeBuilder.buildObject();
        uid.setIsRequired(true);
        uid.setFriendlyName("uid");
        uid.setName("urn:oid:0.9.2342.19200300.100.1.1");
        uid.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(uid);

        RequestedAttribute displayName = RequestAttributeBuilder.buildObject();
        displayName.setIsRequired(true);
        displayName.setFriendlyName("displayName");
        displayName.setName("urn:oid:2.16.840.1.113730.3.1.241");
        displayName.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(displayName);

        RequestedAttribute eduPersonTargetedID = RequestAttributeBuilder.buildObject();
        eduPersonTargetedID.setIsRequired(false);
        eduPersonTargetedID.setFriendlyName("eduPersonTargetedID");
        eduPersonTargetedID.setName("urn:oid:1.3.6.1.4.1.5923.1.1.1.10");
        eduPersonTargetedID.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(eduPersonTargetedID);

        RequestedAttribute eduPersonPrincipalName = RequestAttributeBuilder.buildObject();
        eduPersonPrincipalName.setIsRequired(false);
```

```
        eduPersonPrincipalName.setFriendlyName("eduPersonPrincipalName");
        eduPersonPrincipalName.setName("urn:oid:1.3.6.1.4.1.5923.1.1.1.6");
        eduPersonPrincipalName.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(eduPersonPrincipalName);

        RequestedAttribute organizationName = RequestAttributeBuilder.buildObject();
        organizationName.setIsRequired(false);
        organizationName.setFriendlyName("organizationName");
        organizationName.setName("urn:oid:2.5.4.10");
        organizationName.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(organizationName);

        RequestedAttribute schacHomeOrganization = RequestAttributeBuilder.buildObject();
        schacHomeOrganization.setIsRequired(false);
        schacHomeOrganization.setFriendlyName("schacHomeOrganization");
        schacHomeOrganization.setName("urn:oid:1.3.6.1.4.1.25178.1.2.9");
        schacHomeOrganization.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(schacHomeOrganization);

        RequestedAttribute eduPersonAffiliation = RequestAttributeBuilder.buildObject();
        eduPersonAffiliation.setIsRequired(false);
        eduPersonAffiliation.setFriendlyName("eduPersonAffiliation");
        eduPersonAffiliation.setName("urn:oid:1.3.6.1.4.1.5923.1.1.1.1");
        eduPersonAffiliation.setNameFormat("urn:oasis:names:tc:SAML:2.0:attrname-format:uri");
        service.getRequestAttributes().add(eduPersonAffiliation);

        service.setIndex(2);
        return service;
    }
}
```

3.5 RelayState 의 적용

Spring Security SAML extension 은 RelayState 값을 변경할 수 없는 문제가 있다. RelayState 에서 인증 성공 후 redirection 될 URL 등이 포함된다. 인증 성공 후 return 될 URL 주소 등을 임의로 적용하기 위해 SAMLEntryPoint 클래스를 구현한다. SAMLEntryPoint 클래스는 samlEntryPoint bean 의 class 를 오버라이딩한다.

```
<bean id="samlEntryPoint" class="spring.security.saml.SAMLEntryPoint">
   <property name="defaultProfileOptions">
      <bean class="org.springframework.security.saml.websso.WebSSOProfileOptions">
         <property name="includeScoping" value="false" />
      </bean>
   </property>
</bean>
```

다음과 같이 SAMLEntryPoint 클래스를 생성한다.

```
package spring.security.saml;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.opensaml.common.SAMLException;
import org.opensaml.saml2.metadata.AssertionConsumerService;
import org.opensaml.saml2.metadata.SPSSODescriptor;
import org.opensaml.saml2.metadata.provider.MetadataProviderException;
import org.opensaml.ws.message.encoder.MessageEncodingException;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.saml.SAMLConstants;
import org.springframework.security.saml.context.SAMLMessageContext;
import org.springframework.security.saml.util.SAMLUtil;
import org.springframework.security.saml.websso.WebSSOProfileOptions;

public class SAMLEntryPoint extends org.springframework.security.saml.SAMLEntryPoint{

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
AuthenticationException e) throws IOException, ServletException {
        String relayStateString = request.getQueryString();
        String relayState="";
        try {
            relayState = relayStateString.substring(relayStateString.indexOf("RelayState"),
relayStateString.length()).replace("RelayState=", "");
            System.out.println(relayState);
        } catch (IndexOutOfBoundsException ex) {
        } catch (NullPointerException ex){
        }

        try {
            SAMLMessageContext context = contextProvider.getLocalAndPeerEntity(request, response);

            if (isECP(context)) {
                initializeECP(context, e);
            } else if (isDiscovery(context)) {
                initializeDiscovery(context);
            } else {
                initializeSSO(context, e, relayState);
            }

        } catch (SAMLException e1) {
            logger.debug("Error initializing entry point", e1);
```

```
            throw new ServletException(e1);
        } catch (MetadataProviderException e1) {
            logger.debug("Error initializing entry point", e1);
            throw new ServletException(e1);
        } catch (MessageEncodingException e1) {
            logger.debug("Error initializing entry point", e1);
            throw new ServletException(e1);
        }

    }

    protected void initializeSSO(SAMLMessageContext context, AuthenticationException e, String relayState)
throws MetadataProviderException, SAMLException, MessageEncodingException {
        // Generate options for the current SSO request
        WebSSOProfileOptions options = getProfileOptions(context, e);

        if(!relayState.isEmpty()){
            options.setRelayState(relayState);
        }

        // Determine the assertionConsumerService to be used
        AssertionConsumerService consumerService = SAMLUtil.getConsumerService((SPSSODescriptor)
context.getLocalEntityRoleMetadata(), options.getAssertionConsumerIndex());

        // HoK WebSSO
        if (SAMLConstants.SAML2_HOK_WEBSSO_PROFILE_URI.equals(consumerService.getBinding())) {
            if (webSSOprofileHoK == null) {
                logger.warn("WebSSO HoK profile was specified to be used, but profile is not configured in
the EntryPoint, HoK will be skipped");
            } else {
                logger.debug("Processing SSO using WebSSO HolderOfKey profile");
                webSSOprofileHoK.sendAuthenticationRequest(context, options);
                samlLogger.log(SAMLConstants.AUTH_N_REQUEST, SAMLConstants.SUCCESS, context);
                return;
            }
        }

        // Ordinary WebSSO
        logger.debug("Processing SSO using WebSSO profile");
        webSSOprofile.sendAuthenticationRequest(context, options);
        samlLogger.log(SAMLConstants.AUTH_N_REQUEST, SAMLConstants.SUCCESS, context);

    }
}
```

위 글 상자의 SAMLEntryPoint 클래스를 이용한다면 RelayState 는 반드시 파라미터 중 가장 마지막에 위치해야 한다.

3.6 전달받은 사용자 속성정보의 저장

Spring Security SAML extension 은 eduPersonTargetedID 값을 해석하지 못한다. 사용자 속성 정보 중 eduPersonTargetedID 값을 해석하고 저장하기 위해 UserAttribute 클래스와 SAMLUserDetailsService 클래스를 생성한다. UserAttribute 클래스는 전달받은 사용자 속성정보를 List 와 HashMap 을 이용해 저장한다.

```
package spring.security.saml;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import org.apache.commons.collections.ListUtils;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;

public class UserAttribute {

    private List<HashMap<String, String>> attributes = null;

    public UserAttribute() {
        attributes = new ArrayList<>();
    }

    public void setAttribute(String key, String value){
        HashMap<String, String> attributeSet = new HashMap<>();
        attributeSet.put(key, value);
        attributes.add(attributeSet);
    }

    public final List<HashMap<String, String>> getAttributes() {
        return ListUtils.unmodifiableList(attributes);
    }
}
```

구현된 SAMLUserDetailsService 에서 UserAttribute 객체를 이용한다. SAMLUserDetailsService 를 securityContext.xml 에 등록한다. samlAuthenticationProvider bean 을 찾은 후 property 를 설정한다.

```
<bean                                                                    id="samlAuthenticationProvider"
class="org.springframework.security.saml.SAMLAuthenticationProvider">
   <property name="forcePrincipalAsString"><value>false</value></property>
   <property name="userDetails" ref="samlUserDetailsService" />
</bean>

<bean id="samlUserDetailsService" class="spring.security.saml.SAMLUserDetailsService">
</bean>
```

SAMLUserDetailsService 는 org.springframework.security.saml.userdetails.SAMLUserDetails Service 를 implement 한다. 구현해야하는 함수는 loadUserBySAML 이다.

```
package spring.security.saml;

import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.opensaml.saml2.core.Attribute;
import org.opensaml.ws.message.encoder.MessageEncodingException;
import org.opensaml.xml.util.XMLHelper;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.saml.SAMLCredential;
import org.springframework.security.saml.util.SAMLUtil;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public                    class                    SAMLUserDetailsService                    implements
org.springframework.security.saml.userdetails.SAMLUserDetailsService{

    private final String eptid_oid="urn:oid:1.3.6.1.4.1.5923.1.1.1.10";
    private final String eptid_fname="eduPersonTargetedID";

    @Override
    public UserAttribute loadUserBySAML(SAMLCredential samlc) throws UsernameNotFoundException {
        UserAttribute uattr = new UserAttribute();

        List<Attribute> attrList = samlc.getAttributes();
        for (Attribute attr : attrList) {
            // Spring Security SAML은 XML 형태로 넘어오는 값(value)를 파싱하지 못함
            // 현재 KAFE에서 정의하는 eduPersonTargetedID(eptid)는 XML 형태의 값을 가짐
            // oid가 eduPersonTargetedID(urn:oid:1.3.6.1.4.1.5923.1.1.1.10)일 경우 SAML Assertion을 직접
              파싱해 값을 구해옴
            // getEduPersonTargetedID 함수가 사용됨
            if(attr.getName().equalsIgnoreCase(eptid_oid) | attr.getName().equalsIgnoreCase(eptid_fname)){
                //oid로 eptid 조회
                String eptid = samlc.getAttributeAsString(eptid_oid);
                //eptid 값이 null이 거나 없을 경우 friendlyName으로 etpid 조회
                if(eptid == null || eptid.isEmpty()){
                    eptid = samlc.getAttributeAsString(eptid_fname);
```

```
                }
                if(eptid == null || eptid.isEmpty()){
                    //oid 및 friendlyName으로 모두 조회 했음에도 eptid를 가져오지 못했을 경우 Assertion에서
                    eptid 파싱
                    uattr.setAttribute(attr.getName(), getEduPersonTargetedID(samlc));
                }else{
                    //oid 및 friendlyName으로 조회된 eptid 값이 있을 경우 그대로 입력
                    uattr.setAttribute(attr.getName(), samlc.getAttributeAsString(attr.getName()));
                }
            }else{
                uattr.setAttribute(attr.getName(), samlc.getAttributeAsString(attr.getName()));
            }
        }
        return uattr;
    }

    private String getEduPersonTargetedID(SAMLCredential samlCredential) {
        String eduPersonTargetedID = "";

        try {
            Element e = SAMLUtil.marshallMessage(samlCredential.getAuthenticationAssertion());
            NodeList attrStatementList = e.getElementsByTagName("saml:AttributeStatement");
            for (int i = 0; i < attrStatementList.getLength(); i++) {
                Node attrStatement = attrStatementList.item(i);
                NodeList attrSetList = attrStatement.getChildNodes();
                for (int j = 0; j < attrSetList.getLength(); j++) {
                    Node attrSet = attrSetList.item(j);
                    i                                                                                    f
(attrSet.getAttributes().getNamedItem("Name").getTextContent().equalsIgnoreCase(eptid_oid) |
attrSet.getAttributes().getNamedItem("Name").getTextContent().equalsIgnoreCase(eptid_fname) ) {
                        eduPersonTargetedID = attrSet.getTextContent();
                    }
                }
            }
System.out.println(XMLHelper.nodeToString(SAMLUtil.marshallMessage(samlCredential.getAuthenticationAsser
tion())));
        } catch (MessageEncodingException ex) {
            Logger.getLogger(SAMLUserDetailsService.class.getName()).log(Level.SEVERE, null, ex);
        }
        return eduPersonTargetedID;
    }
}
```

인증된 사용자의 속성 정보는 다음 코드를 이용해 획득할 수 있다.

```
Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
UserAttribute uattrClass = (UserAttribute)authentication.getPrincipal();
```

사용자 속성값은 OID 형태 또는 friendlyName 으로 표기할 수 있다. KAFE 에 참여하는 ID 제공자들은 OID 형태를 이용한다. OID 형태를 friendlyName 으로 변환하는 부분은 KAFE 에서 제공하는 소스코드를 참조한다.

3.7 ID 제공자 선택화면 생성 및 로그아웃

다수의 ID 제공자가 서비스 제공자와 연동될 경우, 서비스 제공자는 “ID 제공자 선택” 화면을 제공해야 한다. “ID 제공자 선택”을 discovery service 라고 한다. securityContex.xml 파일에 org.springframework.security.saml.SAMLDiscovery bean 을 찾아 idpSelectionPath 속성값으로 ID 제공자 선택 페이지의 경로를 지정한다.

```
<bean id=“samlIDPDiscovery” class=“org.springframework.security.saml.SAMLDiscovery”>
  <property name=“idpSelectionPath” value=“/auth/login.jsp” />
</bean>
```

login.jsp 는 사용자가 ID 제공자를 선택하기 위한 웹 페이지이다. login.jsp 에서는 로그인 URL 을 생성하고 사용자가 ID 제공자를 선택하면 해당 ID 제공자로 redirect 시키는 기능을 포함해야 한다. 로그인 URL 을 생성하는 방법은 아래 코드를 참고한다.

```
<%
  WebApplicationContext context =
WebApplicationContextUtils.getWebApplicationContext(getServletConfig().getServletContext());
  MetadataManager mm = context.getBean("metadata", MetadataManager.class);
  // SP에 다수의 IdP가 연결될 수 있으므로 Set 형태로 IdP EntityID를 받아옴
  Set<String> idps = mm.getIDPEntityNames();

  String idpDiscoReturnURL = String.valueOf(request.getAttribute("idpDiscoReturnURL"));
  String idpDiscoReturnParam = String.valueOf(request.getAttribute("idpDiscoReturnParam"));
  String idpLoginBaseURL = idpDiscoReturnURL + "&" + idpDiscoReturnParam + "=";

  // IdP별 로그인 주소를 생성
  for (String idp : idps) {
    //loginURL = 로그인 주소
    //RelayState를 사용할 경우 로그인 URL의 가장 뒤에 작성되어야 함.
    //RelayState의 형태는 다음과 같음 ->
RelayState=/MY/LOGIN/SERVLET?returnTo=YOUR_RETURN_TO_ADDR&YOUR_PARAM1=PARAM1....
    String loginURL = idpLoginBaseURL + idp +
"&RelayState=/myauth/login?returnTo=returnToaddr&param1=p1&parma2=p2";
  }
%>
```

KAFE 에서 제공하는 Web app 에서는 로그아웃 시 다음 URL 로 redirect 한다. 자세한 사항은 소스코드를 참조한다. Single Logout 은 ID 제공자에게 명시적인 로그아웃 메시지를 전달한다. Local Logout 는 명시적 로그아웃 메시지를 전달하지 않고 서비스 제공자에서만 로그아웃 된다.

```
Local Logout : /saml/logout?local=ture
Single Logout : /saml/logout
```

3.8 인증서 관리

SAML 에서는 self-signed 인증서를 사용한다(주의: TLS 인증서는 공인인증서를 이용해야 한다). SHA1 은 보안문제로 사용을 금지하며 SHA256 의 이용을 권장하고 인증서 유효기간은 10 년으로 설정한다. 먼저 사용할 KeyStore 를 securityContext.xml 에 등록한다.

org.springframework.security.saml.key.JSKeyManager bean 을 검색해 다음과 같이 설정한다.

```
<bean id="keyManager" class="org.springframework.security.saml.key.JKSKeyManager">
  <constructor-arg value="classpath:security/samlKeystore.jks" />
  <constructor-arg value="java.lang.String" value="KeyStore 암호" />
  <constructor-arg>
    <map>
      <enter key="apollo" value="apollo123" />
      <entry key="myspcert" value="인증서 암호" />
    </map>
  </constructor-arg>
  <constructor-arg type="java.lang.String" value="myspcert" />
</bean>
```

위 글박스의 내용은 다음과 같다.

라인	내용
<constructor-arg value="classpath:security/samlKeystore.jks" />	KeyStore의 경로
<constructor-arg value="java.lang.String" value="KeyStore 암호" />	KeyStore의 암호
<entry key="myspcert" value="인증서 암호" />	등록한 인증서의 별칭과 암호
<constructor-arg type="java.lang.String" value="myspcert" />	default 인증서의 별칭

위 내용을 바탕으로 인증서를 생성하고 KeyStore 에 등록한다.

```
keytool -genkey -alias myspcert -validity 3650 -keyalg RSA -sigalg SHA256withRSA -keysize 2048
-keystore samlKeystore.jks -keypass [인증서 암호] -storepass [KeyStore 암호] -dname "CN=ㅅ
testsp.kreonet.net,OU=DEPT_NAME,O=ORGNAME,L=Location, S=City, C=KR"

예) keytool -genkey -alias myspcert -validity 3650 -keyalg RSA -sigalg SHA256withRSA -keysize 2048
-keystore samlKeystore.jks -keypass myspcertpass -storepass samlkeystorepass -dname
"CN=testsp.kroenet.net,OU=COREEN,O=KISTI,L=Yuseong,S=Daejeon,C=KR"
```

서비스 제공자가 KAFE 에서 제공하는 Federation metadata 를 이용한다면 Federation metadata 를 생성할 때 이용된 인증서를 KeyStore 에 등록해야 한다. KAFE 에서 제공하는 Federation metadata 의 인증서는 https://fedinfo.kreonet.net/kafe-fed.crt 에서 다운로드 받을 수 있다. KAFE 인증서를 다운로드 받아 KeyStore 에 등록한다.

```
keytool -importcert -keystore samlKeystore.jks -storepass [KeyStore 암호] -alias kafe-fed -file kafe-fed.crt
ex) keytool -importcert -keystore samlKeystore.jks -storepass mypassword -alias kafe-fed -file kafe-fed.crt
```

3.9 기타

securityContext.xml 에 추가 설정이 필요하다.

기본 redirect 주소(RelayState 에 return URL 을 설정하지 않았을 경우 redirect 되는 주소)를 설정한다. 로그인에 성공한 사용자를 redirect 할 주소이다.

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

로그인에 실패했을 경우 redirect 될 주소를 설정한다.

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

로그아웃에 성공했을 때 rediret 될 주소를 설정한다.

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

제 4 장 예제 프로젝트

KAFE 에서 제공하는 예제 응용은 Apache Tomcat 8.0.36 에서 검증되었다. 전달받은 Web app, Spring-Security-SAML-1.0-SNAPSHOT.war 파일의 압축을 푼다. WEB-INF/securityContext.xml 파일을 열어 아래 파란색으로 표시된 부분을 수정한다.

```
<bean id="metadataGeneratorFilter"
class="org.springframework.security.saml.metadata.MetadataGeneratorFilter">
  .....
  <!-- tomcat 기본설치시 port는 8080으로 설정된다 -->
```

```
<property value="https://URL:TOMCAT_PORT/Spring-Security-SAML-1.0-SNAPSHOT" name="entityBaseURL" />
.....
```

수정이 완료되었으면 Web App 을 zip 으로 다시 압축한다. 압축한 후 확장자를 war 로 변경한다. 예제 응용을 배포할 서버에 Apache Tomcat 을 다운로드(http://tomcat.apache.org/download-80.cgi)받고 압축을 푼다.

tomcat-users.xml 파일을 수정해 manager-gui 권한을 가진 사용자를 생성한다. 아래 파란색으로 표시된 두 라인을 추가한다. 추가 후 tomcat 을 실행한다.

```
~# vim /TOMCAT/PATH/conf/tomcat-users.xml
<tomcat-users xmlns=.....>
    ....
    <role rolename="manager-gui"/>
    <user username="USER_NAME" password="USER_PASSWORD" roles="manager-gui,admin-gui"/>
</tomcat-users>
```

https://URL:TOMCAT_PORT/manager 에 접속한다. 위 글박스의 USER_NAME 과 USER_PASSWORD 를 이용해 로그인한다. WAR to deploy 항목에서 예제 Web App 를 선택하고 Deploy 한다.

※ 배포되는 소스코드에 Web App용 인증서와 KAFE 인증서가 함께 KeyStore에 등록되어 있다. 인증서를 KeyStore에 등록하지 않아도 정상 동작한다.

Deploy 가 완료되면 https://URL:TOMCAT_PORT/Spring-Security-SAML-1.0-SNAPSHOT 에 접속한다. “이곳을 클릭해 로그인 하십시오”를 클릭한다. 드롭박스에 있는 IDP 중 “https://testidp.kreonet.net/idp/simplesamlphp”를 선택한다. 사용자 아이디는 student 이며 비밀번호는 student1234 이다.

IDP 선택 시 Metadata Notfound 에러가 발생하면 다음 절차를 따른다.

1. 예제 응용의 첫 페이지에서 “이곳을 클릭해 SP 메타데이터를 다운로드 하십시오”를 클릭해 메타데이터를 다운로드 한다.

2. https://testidp.kreonet.net 으로 접속한다.

3. “메타데이터 등록” 탭에서 “파일 선택”을 클릭한다. 1 번에서 다운로드 한 메타데이터 파일을 선택한 후 “Upload” 버튼을 클릭한다.

4. 사용자 인증을 다시 시도한다.

Java with OIOSAML

제 1 장 개발환경

본 매뉴얼은 다음과 환경에서 활용할 수 있다.

- Java 기반의 웹 응용에 SAML 기능을 부여하고 하는 경우

본 문서의 내용은 다음과 같은 환경에서 검증되었다.

항목	비고
OS	Windows 7 Pro 64bit
Java	Oracle Java 1.7.0
Web container	Tomcat 8.0.15
IDE	Netbeans 8.0.2
oiosaml.java 버전	21188
OpenSAML 2 버전	2.6.6.

본 문서 내용의 참조 사이트는 다음과 같다.

항목	비고
oiosaml.java	https://svn.softwareborsen.dk/oiosaml.java/sp/trunk/docs/intro.html https://digitaliser.dk/group/42063/resources
oiosaml.java Discovery 서비스 소스 다운로드	https://svn.softwareborsen.dk/oiosaml.java/discovery/trunk/src/dk/itst/oiosaml/discovery/service/
OpenSAML	https://wiki.shibboleth.net/confluence/display/OpenSAML/Home/

기존 웹 응용 또는 신규 웹 응용에 KAFE 에서 제공하는 java template 를 수정없이 적용하고자 할 때는 제 4 장과 제 5 장의 내용을 참조한다.

제 2 장 oiosaml.java 데모 설치

2.1 oiosaml.java-demo 설치

1. oiosaml.java 21181(버전 중요)을 다운로드 받은 후 압축을 푼다. oiosaml.java-demo-21188.war 를 다시 압축 해제한다.

2. Netbeans 를 실행한다.

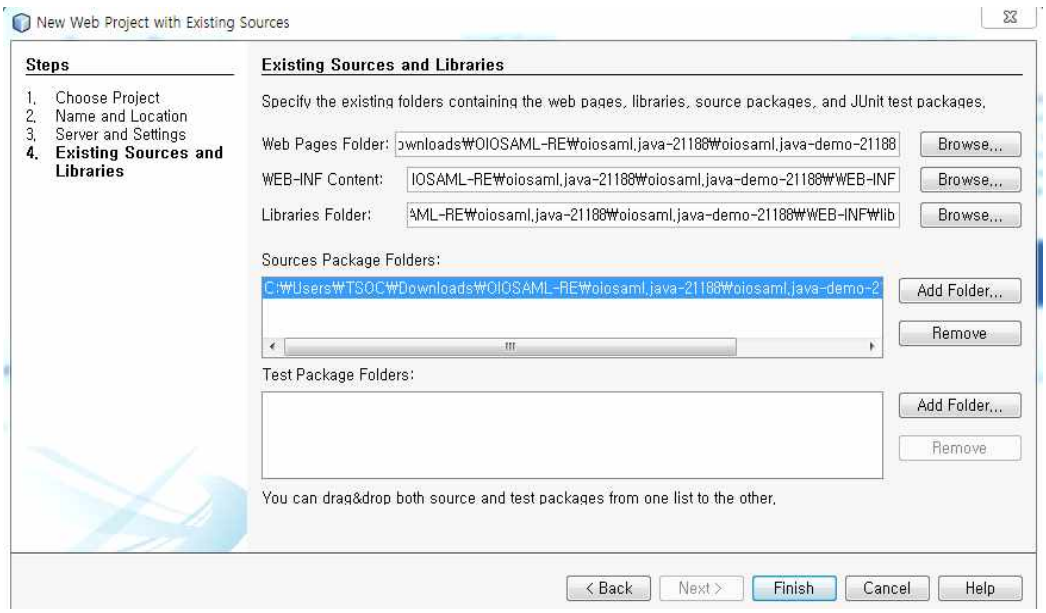
3. New Project -> Java Web -> Web Application with Existing Sources 를 선택한다.

4. 로케이션 / 프로젝트 이름 / 프로젝트 폴더를 지정한 후 Next 를 클릭한다.

ex) (소스)로케이션 : C:\Users\...\oiosaml.java-21188\oiosaml.java-demo-21188
프로젝트 이름 : OIOSAML-DEMO
프로젝트 폴더 : C:\Users\TSOC\Downloads\NetbeansProject\OIOSAML

5. 서버로 Apache Tomcat 8.0.15.0 / JAVA EE 7 Web 을 선택 후 Next 를 클릭한다.

6. 소스 로케이션을 기반으로 WEB-INF 폴더와 Libraries 폴더가 자동으로 지정된다. Finish 를 클릭한다.



7. OIOSAML-DEMO 프로젝트를 우클릭하여 Run 을 클릭한다. 웹 브라우저를 실행해 OIOSAML 데모페이지를 확인한다.

8. oiosaml.java 21188 에 포함된 라이브러리를 최신 버전으로 업데이트 한다.

1) OIOSAML-DEMO 프로젝트의 Libraries 폴더에서 JDK 와 Apache Tomcat 을 제외한 모든 jar 파일을 선택한다. 우 클릭한 후 Remove 를 선택한다.

2) Libraries 폴더를 우클릭하여 Add JAR/Folder 를 선택한다. oiosaml.java-demo-21188/lib 폴더와 opensaml 2.6.6/lib 폴더에서 아래 표에 있는(15 번 제외) 라이브러리를 추가한다. 15 번 oiosaml.java-21188 은 소스를 수정해야 하므로 라이브러리에 추가하지 않는다.

순번	비고
1	commons-codec-1.7
2	commons-collections-3.2.1
3	commons-configuration-commons-configuration
4	commons-digester-commons-digester
5	commons-fileupload-commons-fileupload
6	commons-httpclient-3.1
7	commons-io-commons-io
8	commons-lang-2.6
9	commons-logging-commons-logging
10	esapi-2.0.1
11	joda-time-2.2
12	log4j-1.2.17
13	log4j-over-slf4j-1.7.5
14	not-yet-commons-ssl-0.3.9
15	oiosaml.java-21188

16	opensaml-2.6.6
17	openws-1.5.6
18	org.apache.santuario-xmlsec
19	org.bouncycastle-bcpkix-jdk15on
20	org.bouncycastle-bcprov-jdk15on
21	org.fishwife-jrugged-core
22	velocity-1.7
23	xalan-serializer
24	xalan-xalan
25	xerces-xercesImpl
26	xmlsec-1.5.7
27	xmltooling-1.4.6

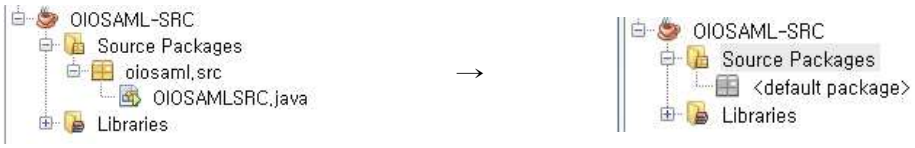
9. oiosaml.java 라이브러리 수정

1) SAML 메타데이터의 signature 검증 부분에서 오류가 발생한다. signature 검증하는 코드를 주석처리한다. ※ 향후 signature 검증 오류를 해소할 수 있는 방안을 찾아야 한다.

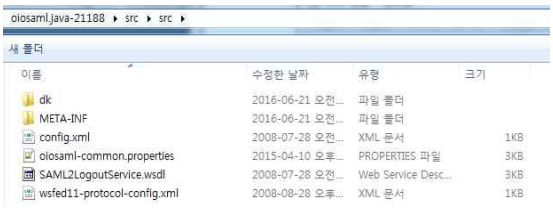
(1) Netbeans 에서 File → New Project → Java → Java Application 을 선택

(2) 프로젝트 이름을 지정하고 Finish 를 클릭

(3) 생성된 프로젝트의 source packages 폴더에 자동으로 생성된 모든 패키지를 삭제 삭제할 경우 아래 그림과 같이 <default package>로 표시 됨



(4) oiosaml.java-21188/src/src 폴더의 모든 파일을 복사. 생성한 자바 프로젝트의 Source packages 를 우클릭하고 붙여 넣기 함



(5) Netbeans 에서 생성한 프로젝트의 Libraries 를 우 클릭하고 Add JAR/Folder 를 클릭.
oiosaml.java-21188/lib 폴더를 선택해 모든 라이브러리 추가.

(6) Libraries 를 우클릭하여 Add JAR/Folder 를 선택. Tomcat 경로에 있는 lib 폴더(ex. C:\Program Files\Apache Software Foundation\Apache Tomcat 8.0.15\lib)로 이동.
servlet-api.jar 를 선택해 라이브러리에 추가.

(7) Netbeans에서 다음 파일을 수정

```
/* package: dk.itst.oiosaml.sp.model
  class: OIOSamlObject
  Method: verifySignature */
181             try {
182                     //주석처리 profileValidator.validate(signature);
183             }
```

```
/* package: dk.itst.oiosaml.sp.model
  class: OIOLogoutResponse
  Method: validator */
// 222, 224 두 라인만 주석처리
220 boolean valid = false;
221             for (PublicKey key : keys) {
222             // 주석처리      if      (Utils.verifySignature(signature,      queryString,
Constants.SAML_SAMLRESPONSE, key)) {
223                             valid = true; log.debug("Signiture validation skip!");
224             //주석처리      }
225             }
```

(8) Netbeans 에서 프로젝트 이름을 우 클릭하여 Properties 를 선택. Build 의 Packaging 항목으로 이동한 후 Compress JAR File 에 체크/Copy Dependent Liabraries 에 체크 후 Ok 클릭.

(9) 생성한 프로젝트 이름을 OIOSAML-DEMO 프로젝트의 라이브러리에 추가. OIOSAML-DEMO 의 Libraries 를 우 클릭한 후 Add Project 를 클릭. 생성한 프로젝트를 선택하고 Add Project AAR Files 를 클릭

(10) 생성한 프로젝트를 우클릭하고 Clean and Build 를 선택. OIOSAML-DEMO 프로젝트를 우클릭하고 Clean and Build 를 선택.

(11) OIOSAML-DEMO 프로젝트를 우 클릭하여 Run 실행. OIOSAML 데모 페이지가 열리면 Page requiring login 버튼을 클릭

2.2 IDP MetaRefresh 기능 추가

KAFE 에서 제공하는 SAML 메타데이터는 주기적으로 업데이트되기 때문에 서비스 제공자와 ID 제공자도 주기적으로 다운로드받아 시스템에 반영해야 한다.

oiosaml.java 는 웹 응용이 실행될 때(load-on-startup/Servlet init) IDP 와 SP 의 메타데이터 정보를 읽어온다. 웹 응용이나 Tomcat 을 재시작 하지 않고 웹 응용이 메타데이터를 reload 할 수 있도록 oiosaml.java 에 기능이 추가되었다. SP 의 메타데이터가 업데이트되었을 때는 반드시 웹 응용을 재시작해야 한다.

Netbeans 에서 다음 파일을 수정한다.

```
/* package: dk.itst.oiosaml.sp.service
  class: DispatcherServlet
  ReloadIdPMetadata() method를 DispatcherServlet Class의 initServlet() method 아래에 추가한다.  */

private void ReloadIdpMetadata(){
  try {
    SAMLConfiguration samlConfiguration = SAMLConfigurationFactory.getConfiguration();
    String protocol = samlConfiguration.getSystemConfiguration().getString(Constants.PROP_PROTOCOL);
    List<XMLObject> descriptors = samlConfiguration.getListOfIdpMetadata();
    idpMetadata = new IdpMetadata(protocol, descriptors.toArray(new EntityDescriptor[descriptors.size()]));
    IdpMetadata.setMetadata(idpMetadata);
    setIdPMetadata(idpMetadata);

    CRLChecker crlChecker = new CRLChecker();
    crlChecker.setAllCertificatesValid(IdpMetadata.getInstance());
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

```
/* package: dk.itst.oiosaml.sp.service
   class: DispatcherServlet
   method: doGet()
   해당 부분을 찾아 아래 빨간색으로 표기된 라인들을 추가한다.
*/
 String action = req.getRequestURI().substring(req.getRequestURI().lastIndexOf("/") + 1);
 Audit.init(req);

 if(action.equals("reloadidpmetadata")){
    ReloadIdpMetadata();
    return ;
 }

 // This is needed if DispatcherServlet isn't protected
 // by the SPFilter
 if (handlers.containsKey(action)) {
    try {
```

```
/* package: dk.itst.oiosaml.sp.service
   class: DispatcherServlet
   method: doPost()
   해당 부분을 찾아 아래 빨간색으로 표기된 라인들을 추가한다.
*/
 String action = req.getRequestURI().substring(req.getRequestURI().lastIndexOf("/") + 1);
 Audit.init(req);

 if(action.equals("reloadidpmetadata")){
    ReloadIdpMetadata();
    return ;
 }

 // This is needed if DispatcherServlet isn't protected
 // by the SPFilter
 if (handlers.containsKey(action)) {
    try {
```

소스코드 수정이 완료되면 Netbeans 에서 프로젝트 이름을 우클릭하여 Properties 를 선택한다. Build 의 Packaging 항목으로 이동하여 Compress JAR File 에 체크하고 Copy Dependent Libraries 를 체크 해제한다. Ok 를 클릭한다.

프로젝트 이름을 OIOSAML-DEMO 프로젝트의 라이브러리에 추가한다. OIOSAML-DEMO 의 Libraries 를 우클릭한 후 Add Project 를 선택한다. 만든 프로젝트를 선택하고 Add Project JAR Files 를 클릭한다.

만든 프로젝트를 우클릭하여 Clean and Build 를 선택한다. 완료되면 OIOSAML-DEMO 프로젝트를 우클릭하고 Clean and Build 를 클릭한다.

OIOSAML-DEMO 프로젝트를 우클릭하여 Run 버튼을 선택한다.

소스코드를 수정한 후 다음 URL 을 호출하면 웹 응용이 IDP 메타데이터를 reload 한다.

```
https://[domain name[:port number]]/saml/reloadidpmetadata
```

2.3 oiosaml.java 설정

1. "configure the system here"를 선택해 설정 페이지로 이동한다.

2. oiosaml 설정은 1 회만 가능하며 설정 정보는 c:\oiosaml-config 에 저장된다. 설정 파일의 위치는 변경 가능하다. 4.2 web.xml 설정을 참고한다. oiosaml 설정을 다시해야 할 경우에는 oiosaml-config 폴더를 삭제하고 프로젝트를 다시 실행시킨다.

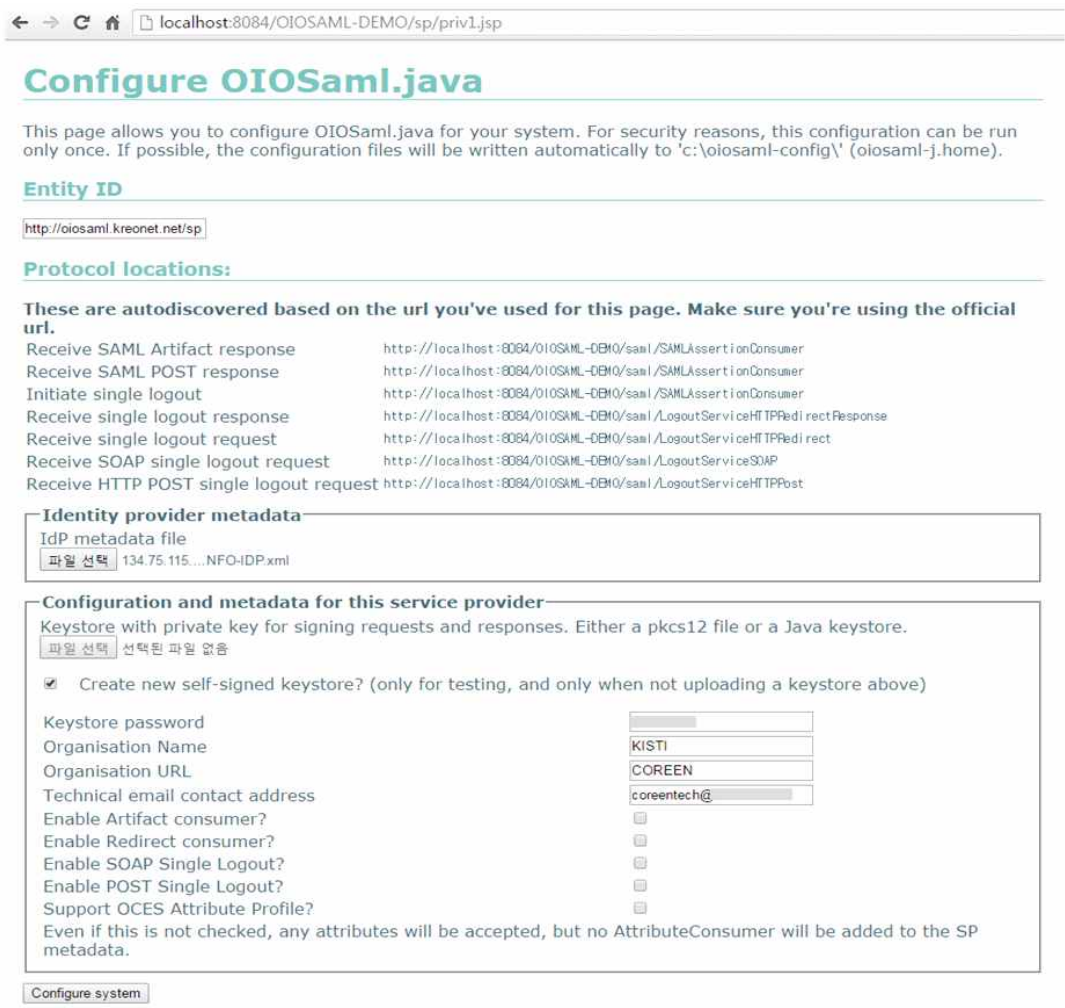
3. 설정 항목은 다음과 같다.

※ OIOSAML 에서 SAML logout 으로 Single Logout 과 Local Logout 을 구분해서 사용하는 경우가 있다. Single Logout 은 ID 제공자에게 명시적인 로그아웃 메시지를 보내는 방식이고 Local Logout 은 ID 제공자에게 명시적인 로그아웃 메시지를 보내지 않는 방식이다. Single Logout 될 경우 재 로그인해야 하지만 Local Logout 인 경우 재 로그인할 필요가 없다.

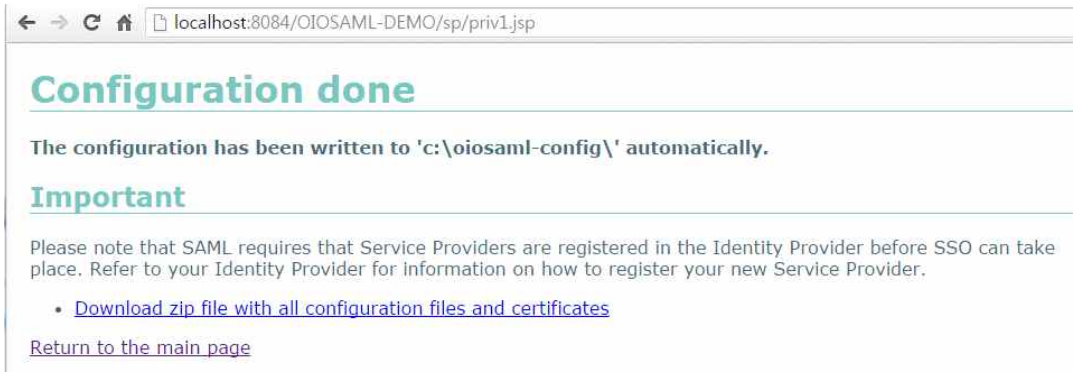
항목	설명	비고
Entity ID	서비스 제공자의 Entity ID Service provider의 EntityID는 https://[domain name]/sp/oiosaml의 형태 임 (e.g., https://example.com/sp/oiosaml)	필수
Identity Provider	ID 제공자의 메타데이터를 XML 형태로 입력	필수

metadata	※ 다수의 ID 제공자를 이용할 경우에는 oiosaml-discovery 서비스를 이용해야 함	
SP 인증서(pkcs12)	반드시 selfsigned 인증서를 이용해야 함 Create new self-signed keystore를 체크한 후 각 항목에 값을 입력. 기존에 사용하던 인증서가 있을 경우 pkcs12 형태로 변경하여 사용	필수
Enable Artifact consumer?	SAML 2.0 HTTP Artifact Bidning을 사용할 경우	
Enable Redirect consumer?	Redirect consumer를 이용	
Enable SOAP Single logout?	SOAP으로 SLO을 호출 가능	
Enable POST Single logout?	HTTP POST로 로그아웃을 호출	
Support OCES Attribute Profile?	DNITA에서 정의하는 OCES 속성 사용 시	

설정 웹 페이지는 다음과 같다.



4. 설정이 완료되면 아래와 같은 페이지가 나타난다.



5. 다음 사항을 추가 설정한다.

1) CLR/OCSP(Online Certificate Status Protocol) 기능의 비활성화: OCSP 는 전자서명에 사용된 인증서의 폐지 상태를 확인하는 프로토콜이다. KAFE 는 CLR/OCSP 기능을 이용하지 않는다. C:\oiosaml-config 의 oiosaml-sp.properties 파일을 열어 다음 설정을 추가한다.

```
oiosaml-sp.crl.period=0
```

2) 메타데이터의 End Point 주소를 자신의 URL 혹은 IP 주소로 수정한다. 설정 페이지를 통해 생성되는 SP 의 SingleLogoutService 및 AssertionConsumerService 주소는 localhost 로 설정된다. C:\oiosaml-config\metadata\SP 의 SPMetadata.xml 을 열어 localhost 로 설정된 부분을 자신의 환경(IP or Domain Name)에 맞게 수정한다. OIOSAML-DEMO 웹 서비스를 다시 빌드 하고 실행(Clean and Build -> Run)시킨다.

3) 메타데이터에 AttributeConsumerService 를 추가하고 NameIDForamt 을 변경한다. SP 의 메타데이터에 필요한 사용자 Attribute 정보를 추가한다. c:\oiosaml-config\metadata\SP\SPMetadata.xml 파일을 열고 <md:AssertionConsumerService ... /> 항목 아래에 다음 내용을 추가한다.

```
<md:AttributeConsumingService index="0">
    <!-- ServiceName 속성의 값(OIOSAML-TEST-SP)을 자신의 웹 응용에 맞게 수정한다. 주로 서비스 이름을 사용한다. (ex : COREEN, facebook 등)-->
    <md:ServiceName xml:lang="en">OIOSAML-TEST-SP</md:ServiceName>
    <md:RequestedAttribute Name="urn:oid:0.9.2342.19200300.100.1.3" isRequired="true" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" FriendlyName="email"/>
    <md:RequestedAttribute Name="urn:oid:0.9.2342.19200300.100.1.1" isRequired="true" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" FriendlyName="uid"/>
    <md:RequestedAttribute Name="urn:oid:2.16.840.1.113730.3.1.241" isRequired="true" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" FriendlyName="displayName"/>
    <md:RequestedAttribute Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" isRequired="false"
```

```
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
FriendlyName="eduPersonTargetedID"/>
    <md:RequestedAttribute Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" isRequired="false"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
FriendlyName="eduPersonPrincipalName"/>
    <md:RequestedAttribute Name="urn:oid:2.5.4.10" isRequired="false"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
FriendlyName="organizationName"/>
    <md:RequestedAttribute Name="urn:oid:1.3.6.1.4.1.25178.1.2.9" isRequired="false"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
FriendlyName="schacHomeOrganization"/>
    <md:RequestedAttribute Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1" isRequired="false"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
FriendlyName="eduPersonAffiliation"/>
</md:AttributeConsumingService>
```

NameIDFormat 을 변경한다.

```
<md:NameIDFormat>...:tc:SAML:1.1:nameid-format:X509SubjectName</md:NameIDFormat>
// 위 라인을 찾아 다음과 같이 변경
<md:NameIDFormat>...:tc:SAML:2.0:nameid-format:persistent</md:NameIDFormat>
```

4) 최종적으로 수정된 메타데이터를 KAFE 에 등록한다.
※ 연동되는 ID 제공자의 메타데이터를 확보해야 한다. ID 제공자와 서비스 제공자를 상호연동하기 위해서는 KAFE staff 의 지원을 받아야 한다. 자세한 사항은 coreen@kreonet.net으로 문의한다.

6. Page requiring login 을 통해 ID 제공자를 선택하고 로그인하면 다음과 같이 속성정보를 확인할 수 있다.

제 3 장 Discovery Service

서비스 제공자가 다수의 ID 제공자와 연동될 때 Discovery Service 를 이용해야 한다. Discovery 서비스는 /oiosaml-config/metadata/IdP 디렉토리에 다수의 ID 제공자 메타데이터가 존재할 경우 자동으로 적용된다.

Discovery Service 사용 시 주의점은 다음과 같다.

* /metadata/IdP 디렉토리에 존재하는 메타데이터 파일 이름은 중요하지 않다.

* KAFE 에서 제공하는 Signed [federation] metadata 를 이용할 수 있다. Federation metadata 에는 서비스 제공자의 메타데이터가 포함되지 않아야 한다.

Discovery 서비스를 적용하기 위해서 다음과 같은 과정이 요구된다.

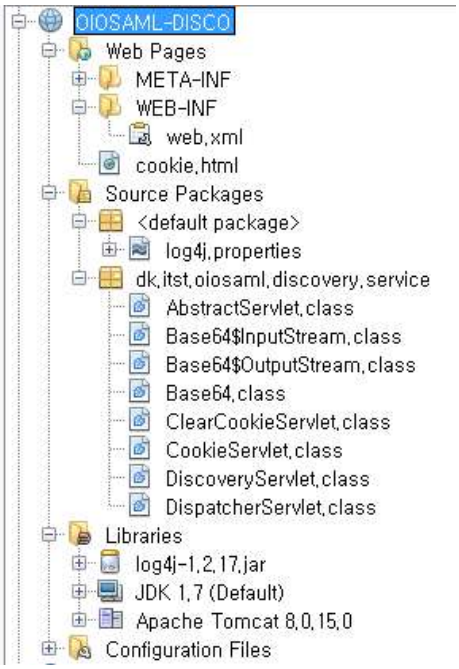
1) Netbeans 에서 File → New Project → Web Application 을 선택한다. 프로젝트 이름과 웹 컨테이너(ex : Tomcat)를 지정하고 Finish 를 클릭한다. 자동으로 생성된 index.html 파일을 삭제한다.

2) oiosaml.java-21188 에 포함되어 있는 oiosaml.java-discovery-1.2.war 파일의 압축을 푼다.

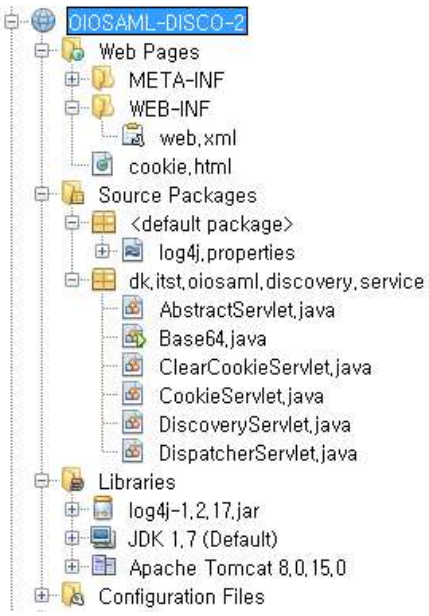
3) oiosaml.java-discovery-1.2/WEB-INF 폴더에 있는 web.xml 을 생성된 프로젝트의 WEB-INF 폴더 아래에 복사한다. oiosaml.java-discovery-1.2 폴더에 있는 cookie.html 은 Web Pages 폴더에 복사한다.

4) oiosaml.java-discovery-1.2/WEB-INF/classes 아래에 있는 모든 파일(dk 폴더, log4j.properties)을 Source Packages 아래에 복사한다. 이때 컴파일 된 class 파일이 아닌 소스(java) 파일을 사용하고 싶을 경우 "2. 홈페이지" 항목의 URL 을 참조하여 discovery service 소스를 다운로드 받아 복사한다.

5) /WEB-INF/lib 에 있는 log4j-1.2.15.jar 파일을 생성된 프로젝트의 Libraries 에 추가한다. 모든 파일이 추가된 프로젝트는 다음과 같은 형태를 갖는다.



class 파일 추가 시



source 파일 추가 시

6) 생성한 프로젝트를 우 클릭하여 빌드한 후 실행(Clean and Build -> Run)한다.

7) OIOSAML-DEMO 프로젝트에 Discovery 서비스 사용을 위한 설정을 추가한다. C:\oiosaml-config\oiosaml-sp.properties 파일에 다음 내용을 추가한다.

```
oiosaml-sp.discovery=DISCOVERY_URL
ex) oiosaml-sp.discovery=http://IP or DomainName:8084/OIOSAML-DISCO-2
// OIOSAML-DISCO-2는 웹 응용의 context root이다.
```

8) OIOSAML-DEMO 프로젝트를 우클릭하여 실행(Run)을 클릭한다. 웹 페이지가 로딩되면 Page requiring login 을 클릭한다. 설정 폴더(C:\oiosaml-config\metadata\IdP) 아래에 IdP 메타데이가 2 개 이상일 경우 자동으로 Discovery 서비스로 연결 된다. 하나의 Federation Metadata 파일에 다수의 IdP Entity Metadata 가 있는 경우에도 동일하게 Discovery 서비스로 연결된다. 정상적으로 실행이 되면 다음과 같은 페이지를 확인 할 수 있다.

제 4 장 Java 웹 응용에 oiosaml.java 적용

Java 응용에 OIOSAML 을 적용하기 위해서 다음과 같은 절차가 필요하다.

- 서버의 NTP 시간 동기화 필수(time.kriss.re.kr 등 이용)

- Federation metadata 의 다운로드 자동화(coreen@kreonet.net에 문의)

- Java 응용에 필수 라이브러리 등록

- web.xml 에 필수 서블릿 및 필터 등록

- oiosaml.java 의 Service Provider 설정(oiosaml-config)

1. oiosaml.java 라이브러리를 Java 응용에 추가

   1) TestJSP 예제 프로젝트의 lib 폴더에 포함된 모든 라이브러리를 Java 응용에 추가한다. TestJSP 는 KAFE 에서 제공하는 Java SAML 예제 패키지이다.

2. 아래 내용을 web.xml 파일에 추가한다.

```
<env-entry>
   <env-entry-name>oiosaml-j.home</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <!-- oiosaml 설정 파일이 저장될 디렉토리 -->
   <env-entry-value>c:\oiosaml-config-TestJSP</env-entry-value>
</env-entry>

<listener>

<listener-class>dk.itst.oiosaml.sp.service.session.SessionDestroyListener</listener-class><
/listener>
   <servlet>
      <servlet-name>SAMLDispatcherServlet</servlet-name>
      <servlet-class>
         dk.itst.oiosaml.sp.service.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <!--
         SAML 사용자 인증 및 Logout을 위한 서블릿
         사용자 인증 요청 : /saml/login 으로 포워딩
         사용자 로그아웃(Single Logout) 요청 : /saml/Logout 으로 포워딩
```

```
    -->
    <servlet-name>SAMLDispatcherServlet</servlet-name>
    <url-pattern>/saml/*</url-pattern>
</servlet-mapping>
<!-- 로그인 필터 이용 -->
<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>dk.itst.oiosaml.sp.service.SPFilter</filter-class>
</filter>
<filter-mapping>
    <!--
        로그인 필터의 URL 패턴은 로그인을 수행하는 서블릿 혹은 JSP 파일의 주소와 매핑
        되어야 함. 본 문서에서는 로그인 절차를 /login 서블릿에서 수행
        ex1) 로그인 절차를 /login 서블릿에서 수행할 경우 url 패턴 :
            <url-pattern>/login/*</url-pattern>
        ex2) 로그인 절차를 /sp/login.jsp에서 수행할 경우 url 패턴 :
            <url-pattern>/sp/*</url-pattern>
    -->
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/login/*</url-pattern>
</filter-mapping>
```

3. Service Provider 설정

※ /saml/configure 와 /saml/metadata 는 서비스 관리자 이외에 접근할 수 없도록 설정해야 한다.

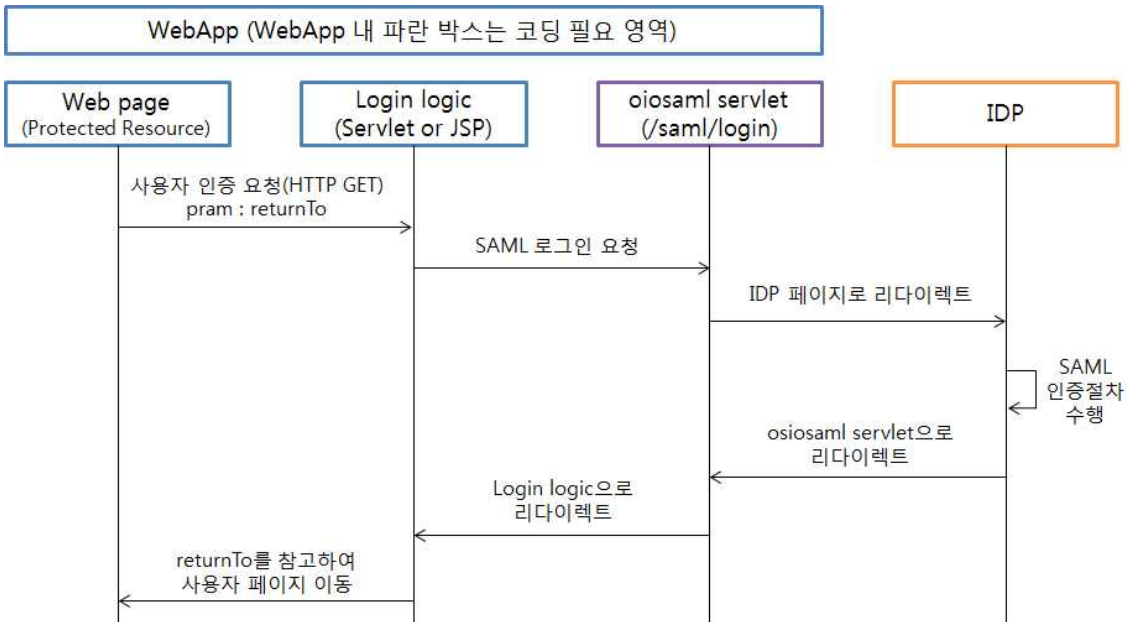
1) “2.2 oiosaml.java 설정”을 참조한다. 기술지원이 필요할 경우 coreen@kreonet.net으로 문의한다.

4. 사용자 인증 요청 예제

TestJSP 에 포함된 예제 코드를 실행시키면 GET, POST 방식의 인증 요청 예제가 포함되어 있다.

1) GET 방식의 인증 요청

GET 방식의 인증 요청 플로우는 다음과 같다.



- WebPage 는 사용자 인증이 필요한 웹 페이지/리소스를 의미한다.

- Login logic 은 SAML 인증을 요청하고 사용자 속성 정보를 전달 받을 서블릿/JSP 를 의미한다.

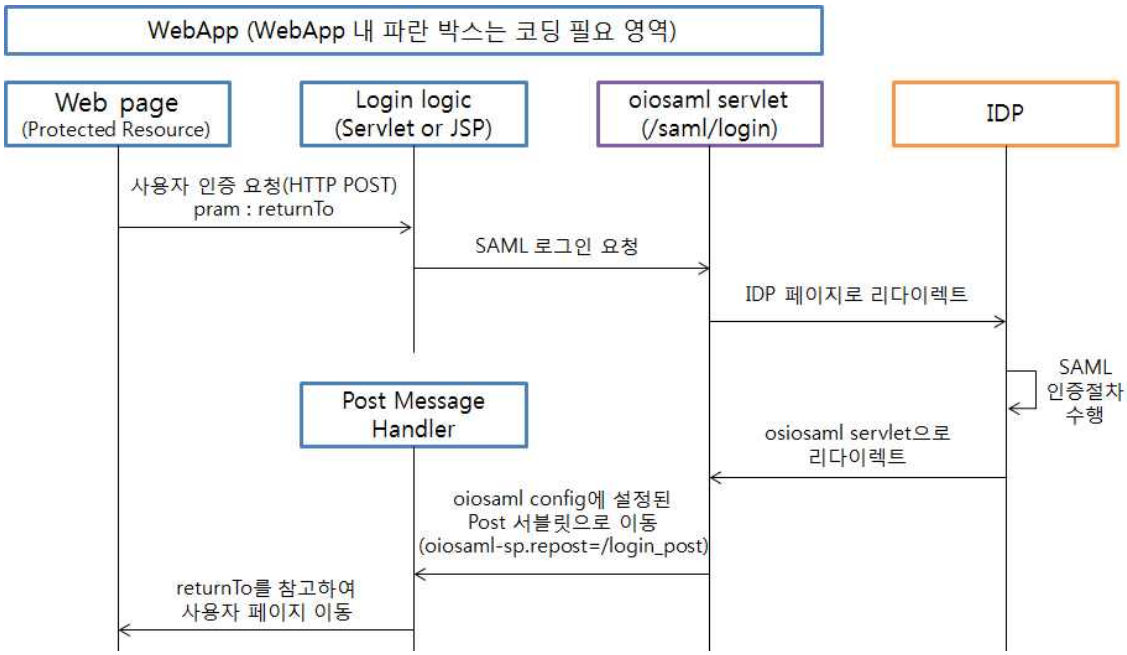
- oiosaml servlet 은 oiosaml.java 의 서블릿을 의미한다.

- 개발자는 Web page 및 Login logic 부분을 구현해야 한다. TestJSP 예제 프로젝트의 소스 파일을 참고하여 구현한다.

Web page : protectectedResource.jsp Login logic : login.java 서블릿

2) POST 방식의 인증 요청

POST 방식의 인증 요청 플로우는 다음과 같다.



- POST 방식의 SAML 인증을 사용하기 위해서는 oiosaml-config 폴더의 oiosaml-sp.properties 파일에 Post Message Handler 설정을 추가해야 한다.

```
oiosaml-sp.repost=SERVLET_URL
ex) oiosaml-sp.repost=/login_post
```

- WebPage 는 사용자 인증이 필요한 웹 페이지/리소스를 의미한다.

- Login logic 은 SAML 인증을 요청할 서블릿/JSP 를 의미한다.

- oiosaml servlet 은 실제 SAML 인증 절차를 수행할 oiosaml.java 의 서블릿을 의미한다.

- Post Message Handler 는 인증 절차 완료 후 사용자 Assertion 정보를 받을 서블릿을 의미한다.

- 개발자는 Web page, Login logic, Post Message Handler 부분을 구현해야 한다. TestJSP 예제 프로젝트의 소스파일을 참고해서 구현한다.

```
Web page : protectectedResource_post.jsp
Login logic : login.java 서블릿
Post Message Handler : login_post.java 서블릿
```

5. 로그아웃 예제

1) Local Logout: Local Logout 은 ID 제공자에 로그아웃 요청을 하지 않고 Java 응용에서만 로그아웃 처리를 한다.

- TestJSP 웹 응용은 UserAssertion 의 isAuthentication 함수로 인증 성공 여부를 확인 후 “isLogin=true” 라는 플래그를 이용해 로그인 성공 여부를 표시한다.

- isLogin 속성의 값을 false 로 변경하고 session 에 UserAssertion 정보가 남아 있다면 삭제 한다.

- local_logout.java 서블릿을 참고하여 구현한다.

2) Single Logout: Single Logout 은 웹 응용에서 로그아웃 처리 후 ID 제공자에게 로그아웃 요청을 전달한다.

- isLogin 속성의 값을 false 로 변경하고 session 에 UserAssertion 정보가 남아 있다면 삭제 한다.

- 다음 URL 로 웹 페이지를 리다이렉트 or 디스패치 한다. URL : /saml/Logout

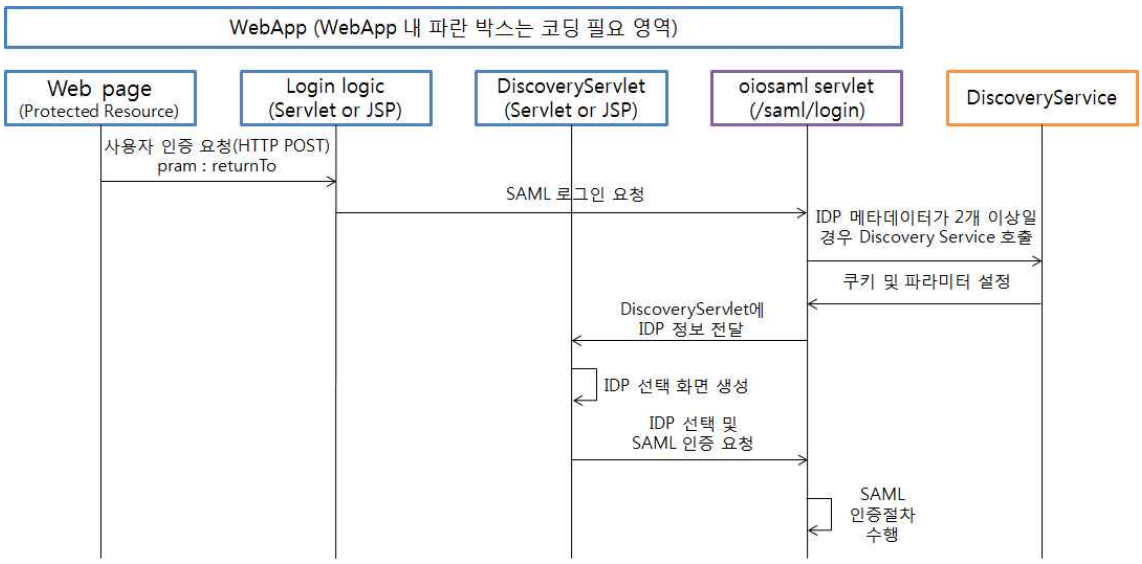
- logout.java 서블릿을 참고하여 구현한다.

6. Discovery service 의 적용

- oiosaml.java-21188.zip 파일에 포함된 “oiosaml.java-discovery-1.2.war”를 웹 컨테이너에 디플로이한다.

- oiosaml-config\metadata\IdP 폴더에 ID 제공자의 메타데이터가 2 개 이상 있을 경우 oiosaml.java 라이브러리에서 자동으로 discovery 서비스를 호출한다.

- Discovery 서비스가 적용될 경우 인증 플로우는 다음 그림과 같다.



SAML 인증절차 수행 후 플로우는 GET 및 POST 인증 방식과 동일

- oiosaml-config 폴더의 oiosaml-sp.properties 파일을 열어 다음 두 설정을 추가한다.

```
oiosaml-sp.discovery={컨테이너에 디플로이한 Discovery 서비스 URL}
ex) oiosaml-sp.discovery=http://IP or DomainName:8084/oiosaml.java-discovery-1.2
oiosaml-sp.discovery.prompt.servlet={IDP 리스트를 전달받을 서블릿}
ex) oiosaml-sp.discovery.prompt.servlet=/DiscoveryServlet
```

- DiscoveryService 는 “oiosaml.java-discovery-1.2.war”로 배포된 웹 서비스를 의미한다.

- DiscoveryServlet 은 DiscoveryService 로부터 IDP 정보를 전달받고 사용자에게 IDP 선택 화면을 제공할 서블릿을 의미한다. 다음 두 소스 파일을 참고하여 구현한다.

```
Discovery Servlet: DiscoveryServlet.java(서블릿), Discovery.jsp(뷰)
```

제 5 장 예제 프로젝트

KAFE 는 Java 로 구현된 웹 응용서비스의 KAFE 참여를 지원하기 위해 SAML Java 예제 코드인 TestJSP.war 를 제공한다. TestJSP.war 는 OIOSAML 라이브러리를 기반으로 작성되었다. TestJSP.war 구동환경은 다음 2 가지 요구조건이 만족되어야 한다.

```
1. NTP 설정
2. PyFF를 이용한 KAFE 페더레이션 메타데이터의 주기적 다운로드 및 MetaRefresh (부록 참고)
```

TestJSP.war 는 다음 환경에서 검증되었다.

```
1. Apache Tomcat 8.0.27
2. Jetty 9.3.9.v20160517
3. GlassFish 4.1
```

TestJSP.war 를 다음과 같은 순서로 설치한다.

1) 파일의 압축을 푼다.

```
~# unzip TestJSP.war -d TestJSP
~# mkdir -p /opt/oiosaml-config/
```

2) Tomcat, GlassFish 의 경우 web.xml 파일을 열어 환경설정 파일들이 위치할 디렉토리를 설정한다.

```
<env-entry>
        <env-entry-name>oiosaml-j.home</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <!-- oiosaml설정 파일을 저장할 디렉토리 -->
        <env-entry-value>/opt/oiosaml-config/TestJSP</env-entry-value>
 </env-entry>
```

3) Jetty 의 경우, 구동환경 설정을 위해 Web.xml 파일을 이용하지 않는다. Jetty 를 이용할 경우 TestJSP/WEB-INF 디렉토리에 jetty-env.xml 파일을 생성한 후 다음과 같이 설정한다.

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN"
   "http://www.eclipse.org/jetty/configure_9_0.dtd">
<Configure id="wac" class="org.eclipse.jetty.webapp.WebAppContext">
  <New id="properties" class="org.eclipse.jetty.plus.jndi.EnvEntry">
        <Arg>oiosaml-j.home</Arg>
        <!-- oiosaml 설정 파일들이 존재할 디렉토리 -->
        <Arg>/opt/oiosaml-config/TestJSP</Arg>
  </New>
</Configure>
```

4) 설정 파일이 저장되지 않는 경우, 설정 파일을 저장할 디렉토리에 쓰기 권한을 부여한다. 권한 부여는 다음 명령을 참고한다.

```
ex) apache일 경우
# chown apache.apache /opt/oiosaml-config
ex) jetty일 경우
# chown jetty.jetty /opt/oiosaml-config
```

5) TestJSP 웹 응용을 웹 컨테이너의 webapps 폴더에 복사한다. 웹 페이지에 접속한다.

```
https://domain_name:port/TestJSP
```

6) 웹 페이지에 접속한 후 OIOSAML 을 환경설정한다. “4.3 Service Provider 설정”을 참조한다.

OIOSAML 의 환경설정화면에서 [부록]의 pyff 로 생성한 메타데이터를 “Identity provider metadata”로 등록한다. pyff 에서 생성한 메타데이터의 파일명을 수정하지 않고 이용해야 한다. 본 장의 설정환경에서 IdP 의 메타데이터는 /opt/oiosaml-config/TestJSP/metadata/IdP 에 저장된다. [부록]의 결과로 /opt/oiosaml-config/TestJSP/metadata/IdP 디렉토리에 위치한 IDP 의 메타데이터가 주기적으로 자동 갱신된다.

7) IDP 메타데이터의 동적 reload

다음 URL 을 wget 이나 curl 을 통해 호출하면 웹 응용이 IDP 메타데이터를 reload 한다.

```
https://domain_name:port/TestJSP/saml/reloadidpmetadata
```

웹 응용에서 IDP 의 메타데이터를 주기적으로 reload 시키기 위해 다음과 같이 cron 을 설정한다.

```
~# crontab -e
0 0,12 * * * wget —spider https://domain_name:port/TestJSP/saml/reloadidpmetadata
```

부록

pyff 는 페더레이션 메타데이터를 다운로드 받아 관리하기 위한 도구이다. simpleSAMLphp 나 Shibboleth 를 이용하지 않는 서비스 제공자는 pyff 를 이용해 페더레이션 메타데이터를 주기적으로 다운로드 받아야 한다.

pyff 는 python 으로 작성된 소프트웨어이며 본 부록의 내용은 Ubuntu 14.04 LTS (64-bit 서버)에서 검증되었다. pyff 의 설치 및 환경설정 방법은 다음과 같다.

기본 패키지를 설치한다.

```
~# apt-get install build-essential python-dev libxml2-dev libxslt1-dev libyaml-dev
```

OS 에 설치된 기존 python 패키지를 이용한다.

```
~# apt-get install python-virtualenv
~# mkdir -p /opt/pyff
~# virtualenv /opt/pyff
```

virtualenv 를 활성화하고 pyff 를 설치한다.

```
~# . /opt/pyff/bin/activate
~# pip install pyFF
~# pip install importlib
```

설치 중에 아래와 관련된 오류가 발생하면 apt-get install lib32z1-dev 를 실행한 후 pyFF 를 다시 설치한다.

```
오류 내용:
/usr/bin/ld: cannot find -lz
~# apt-get install lib32z1-dev
~# pip install pyFF
~# pip install importlib
```

/opt/pyff/certs 디렉토리에 self-signed 인증키와 인증서를 생성한다.

```
~# mkdir -p /opt/pyff/certs
// paste and copy하지 말 것
~# openssl genrsa -out /opt/pyff/certs/sign.key 2048
~# openssl req -key /opt/pyff/certs/sign.key -new -x509 -days 3650 -out /opt/pyff/certs/sign.crt
```

pyFF 는 SHA-1 으로 메타데이터를 서명한다. 서명에 사용되는 알고리즘을 변경하기 위해서 /opt/pyff/lib/python2.7/site-packages/xmlsec/__init__.py 파일을 열어, 'ALGORITHM_SIGNATURE_RSA_SAH1'과 'ALGORITHM_DIGEST_SHA1'을 다음 표를 참조해 변경(SHA256 이용할 것)한다.

알고리즘	ALGORITHM_SIGNATURE_RSA_SHA1	ALGORITHM_DIGEST_SHA1
SHA1	ALGORITHM_SIGNATURE_RSA_SHA1	ALGORITHM_DIGEST_SHA1
SHA256	ALGORITHM_SIGNATURE_RSA_SHA256	ALGORITHM_DIGEST_SHA256
SHA328	ALGORITHM_SIGNATURE_RSA_SHA384	ALGORITHM_DIGEST_SHA384
SHA512	ALGORITHM_SIGNATURE_RSA_SHA512	ALGORITHM_DIGEST_SHA512

/opt/pyff 디렉토리에 kafe-setid.fd 파일을 생성한다. kafe-setid.fd 는 KAFE federation 메타데이타 중 COREEN set.ID 의 메타데이터만 추출해 지정된 디렉토리에 저장한다. 다음은 kafe-setid.fd 파일의 내용이다.

```
- load: "https://fedinfo.kreonet.net/signedmetadata/federation/KAFE-testfed/metadata.xml as kafe-md"
- select: "kafe-md!//md:EntityDescriptor[md:Extensions/mdrpi:RegistrationInfo/@registrationAuthority and md:Extensions/mdrpi:RegistrationInfo/@registrationAuthority='KAFE']"

### All IdPs including KAFE
- fork:
    - select:
        - "kafe-md!//md:EntityDescriptor[@entityID='https://coreen-idp.kreonet.net/idp/simplesamlphp']"
    - xslt:
        stylesheet: tidy.xsl
    - finalize:
        Name: KAFE-IDP
        ID: KAFE-IDP
        cacheDuration: PT5H
        validUntil: P5D
```

```
  - sign:
      key: /opt/pyff/certs/sign.key
      cert: /opt/pyff/certs/sign.crt
  - publish:
      - /opt/oiosaml-config/TestJSP-Jetty/metadata/IdP/IdPMetadata.xml
```

※ 위 /opt/oiosaml-config/TestJSP-Jetty/metadata/IdP/IdPMetadata.xml 는 oiosaml 의 환경설정에 따라 적절히 수정한다.

kafe-setid.fd 에서 이용하는 tidy.xsl 파일의 내용은 다음과 같다. /opt/pyff 디렉토리에 저장한다.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:shibmeta="urn:mace:shibboleth:metadata:1.0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
        xmlns:xi="http://www.w3.org/2001/XInclude"
                xmlns:shibmd="urn:mace:shibboleth:metadata:1.0">

  <xsl:template match="@ID"/>
  <xsl:template match="@validUntil"/>
  <xsl:template match="@cacheDuration"/>

  <xsl:template match="text()|comment()|@*">
    <xsl:copy/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

주기적으로 pyff 가 실행될 수 있도록 crontab 에 등록한다. /opt/pyff/scripts/에 run-pyff.bash 파일을 다음과 같이 생성한다.
※ 아래 /opt/oiosaml-config/TestJSP-Jetty/metadata/IdP/IdPMetadata.xml 는 서버운영 환경에 맞게 설정한다.

```
~# mkdir -p /opt/pyff/scripts
~# nano /opt/pyff/scripts/run-pyff.bash

#!/bin/bash
. /opt/pyff/bin/activate
/opt/pyff/bin/pyff --loglevel=INFO /opt/pyff/kafe-setid.fd
chmod 644 /opt/oiosaml-config/TestJSP-Jetty/metadata/IdP/IdPMetadata.xml
deactivate

~# chmod 755 run-pyff.bash
```

```
~# crontab -e
  // 시간은 적절히 수정
 01 * * * * /opt/pyff/scripts/run-pyff.bash
```