認証

LDAPを用いたパスワード認証

個別のページに移動

Shibboleth IdP 3からは、LDAPモジュールを用いたJAASによるパスワード認証に加えて、直接LDAPを参照するパスワード認証が追加されました。

デフォルトは直接LDAPを参照するパスワード認証です。

直接LDAPを参照するパスワード認証

- conf/ldap.properties
  参照するLDAPにあわせて、Connection properties, SSL configuration, Search DN resolutionのプロパティを設定します。

conf/ldap.properties

```
## Connection properties ##
idp.authn.LDAP.ldapURL                  = ldap://localhost:389
idp.authn.LDAP.useStartTLS              = false
#idp.authn.LDAP.useSSL                  = false
#idp.authn.LDAP.connectTimeout          = 3000



# Search DN resolution, used by anonSearchAuthenticator, bindSearchAuthenticator
# for AD: CN=Users,DC=example,DC=org
idp.authn.LDAP.baseDN                   = ou=people,dc=example,dc=ac,dc=jp
idp.authn.LDAP.subtreeSearch            = true
idp.authn.LDAP.userFilter               = (uid={user})
# bind search configuration
# for AD: idp.authn.LDAP.bindDN=adminuser@domain.com
idp.authn.LDAP.bindDN                   =
idp.authn.LDAP.bindDNCredential         =
```

差分

```
 ## Connection properties ##
-idp.authn.LDAP.ldapURL                  = ldap://localhost:10389
-#idp.authn.LDAP.useStartTLS             = true
+idp.authn.LDAP.ldapURL                  = ldap://localhost:389
+idp.authn.LDAP.useStartTLS              = false
 #idp.authn.LDAP.useSSL                  = false
 #idp.authn.LDAP.connectTimeout          = 3000



 # Search DN resolution, used by anonSearchAuthenticator, bindSearchAuthenticator
 # for AD: CN=Users,DC=example,DC=org
-idp.authn.LDAP.baseDN                   = ou=people,dc=example,dc=org
-#idp.authn.LDAP.subtreeSearch           = false
+idp.authn.LDAP.baseDN                   = ou=people,dc=example,dc=ac,dc=jp
+idp.authn.LDAP.subtreeSearch            = true
 idp.authn.LDAP.userFilter               = (uid={user})
 # bind search configuration
 # for AD: idp.authn.LDAP.bindDN=adminuser@domain.com
-idp.authn.LDAP.bindDN                   = uid=myservice,ou=system
-idp.authn.LDAP.bindDNCredential         = myServicePassword
+idp.authn.LDAP.bindDN                   =
+idp.authn.LDAP.bindDNCredential         =
```

JAASによるパスワード認証

- conf/authn/password-authn-config.xml

  <import resource="jaas-authn-config.xml" />の行をアンコメントして、<import resource="ldap-authn-config.xml" />の行をコメントアウトします。

conf/authn/password-authn-config.xml

```
<!-- Choose an import based on the back-end you want to use. -->
<import resource="jaas-authn-config.xml" />
<!-- <import resource="krb5-authn-config.xml" /> -->
<!-- <import resource="ldap-authn-config.xml" /> -->
```

- conf/authn/jaas.config

  参照するLDAPにあわせて、org.ldaptive.jaas.LdapLoginModule required以降の行を設定します。

conf/authn/jaas.config

```
ShibUserPassAuth {
    /*
        com.sun.security.auth.module.Krb5LoginModule required;
        */
    org.ldaptive.jaas.LdapLoginModule required
      ldapUrl="ldap://localhost"
      baseDn="ou=people,dc=example,dc=ac,dc=jp"
      ssl="false"
      userFilter="uid={user}"
      subtreeSearch="true"
      ;
};
```

LDAPサーバにStartTLSで接続する方法（LDAPサーバがCentOS 6の場合）

個別のページに移動

CentOS 6 標準のopenldap-serversパッケージでLDAPサーバを構築した環境において、IdPからLDAPサーバにStartTLSで接続する設定について記載します。

LDAPサーバはホスト名 ldaptest1.gakunin.nii.ac.jp として説明します。 LDAPプロキシサーバ：複数台LDAPサーバ向けのLDAPプロキシサーバ設定方法 の「LDAPサーバ設定(ldaptest1)」を参考に OpenLDAP の設定を行ってください。この説明で利用する証明書の情報は「LDAPサーバ設定(ldaptest1)」の設定に準じます。

> ⚠ 上記資料はCentOS 5系で記載されたものであるため、利用するバージョンに合わせて適宜読み替える必要があります。

IdPでは ldap.properties, attribute-resolver.xml に下記の設定を行います。例としてLDAPサーバのCA証明書は /etc/pki/tls/certs/gakuninca.pem として配置しています。

/opt/shibboleth-idp/conf/ldap.properties の設定

```
(省略)
## Connection properties ##
idp.authn.LDAP.ldapURL                  = ldap://ldaptest1.gakunin.nii.ac.jp
idp.authn.LDAP.useStartTLS              = true
#idp.authn.LDAP.useSSL                  = false
#idp.authn.LDAP.connectTimeout          = 3000

## SSL configuration, either jvmTrust, certificateTrust, or keyStoreTrust
idp.authn.LDAP.sslConfig                = certificateTrust  ← アンコメント
## If using certificateTrust above, set to the trusted certificate's path
idp.authn.LDAP.trustCertificates        = /etc/pki/tls/certs/gakuninca.pem
## If using keyStoreTrust above, set to the truststore path
idp.authn.LDAP.trustStore               = %{idp.home}/credentials/ldap-server.truststore

(省略)
```

/opt/shibboleth-idp/conf/attribute-resolver.xml の設定

```
    <DataConnector id="myLDAP" xsi:type="LDAPDirectory"
        ldapURL="%{idp.attribute.resolver.LDAP.ldapURL}"
        baseDN="%{idp.attribute.resolver.LDAP.baseDN}"
        principal="%{idp.attribute.resolver.LDAP.bindDN}"
        principalCredential="%{idp.attribute.resolver.LDAP.bindDNCredential}"
        useStartTLS="%{idp.attribute.resolver.LDAP.useStartTLS:true}"
        connectTimeout="%{idp.attribute.resolver.LDAP.connectTimeout}"
        responseTimeout="%{idp.attribute.resolver.LDAP.responseTimeout}"
        noResultIsError="%{idp.attribute.resolver.LDAP.noResultIsError:true}"
        trustFile="%{idp.attribute.resolver.LDAP.trustCertificates}">

        ↑上記のように > の直前に挿入してください
```

/opt/shibboleth-idp/conf/attribute-resolver.xml の設定 (Shibboleth IdP 3.2.xおよびそれ以前の場合)

```
   <resolver:DataConnector id="myLDAP" xsi:type="dc:LDAPDirectory"
(省略)
        ↓以下の3行を</resolver:DataConnector>の直前に挿入してください
        <dc:StartTLSTrustCredential id="LDAPtoIdPCredential" xsi:type="sec:X509ResourceBacked">
            <sec:Certificate>%{idp.attribute.resolver.LDAP.trustCertificates}</sec:Certificate>
        </dc:StartTLSTrustCredential>
   </resolver:DataConnector>
```

⚠ IdPバージョン2向けのattribute-resolver.xmlはldap.propertiesを参照しないため齟齬が発生する恐れがあります。最新のattribute-resolverテンプレートを使用するようにしてください。

ⓘ LDAP Data Connectorでは、 idp.authn.LDAP.sslConfigはcertificateTrustのみ使用可能です。

複数台のLDAPサーバを参照するための方法

個別のページに移動

ここでは2つの方法をご紹介します。

1. 複数台LDAPサーバ向けのLDAPプロキシサーバを使用する
2. IdPの設定ファイルで複数台LDAPサーバを指定する

1. 複数台LDAPサーバ向けのLDAPプロキシサーバを使用する

複数台のLDAPサーバ向けにLDAPプロキシサーバを設置し、IdPにはLDAPプロキシサーバを参照させます。
使用するLDAPプロキシサーバの設定方法について、まとめられた資料があります。資料はCentOS 5系で記載されたものであるため、利用するバージョンに合わせて適宜読み替える必要があります。

⚠ プロキシサーバを構築する際、ログイン画面で入力するID（Shibboleth内部ではprincipalと表現されます）について、同一のIDが複数のLDAPツリー上に存在しないことを確認してください。同一のIDが存在する場合には、属性取得で問題が発生します。uidがこの条件を満たさない場合は、メールアドレスや学籍番号・教職員番号等、他のLDAP属性を使うことを検討してください。

2. IdPの設定ファイルで複数台LDAPサーバを指定する

IdPの設定で複数のLDAPツリー或いは複数のLDAPサーバを参照する例がShibboleth Wiki:LDAPAuthnConfigurationの"DNResolution"の項にあります。最初の例（"Single Directory with multiple branches"の"Extensible Matching"）はLDAPサーバが一台のみで検索すべき複数のLDAPツリーがサブツリーの関係にある環境向けで、ldap.propertiesのみの変更で対応が可能です。idp.authn.LDAP.baseDNのサブツリーを検索するようidp.authn.LDAP.subtreeSearchとidp.authn.LDAP.userFilterを設定します。

- LDAPサーバが一台であり複数のLDAPツリーがサブツリーの関係にある場合の例

/opt/shibboleth-idp/conf/ldap.properties の設定例

```
# Search DN resolution, used by anonSearchAuthenticator, bindSearchAuthenticator
# for AD: CN=Users,DC=example,DC=org
idp.authn.LDAP.baseDN = o=test_o,dc=ac,c=JP
idp.authn.LDAP.subtreeSearch = true ← trueを設定
idp.authn.LDAP.userFilter = (&(|(ou:dn:=Test Unit1)(ou:dn:=technology))(uid={user})) ← o=test_o,dc=ac,c=JPのサブツリー、
ou=Test Unit1とou=technologyを検索するよう設定
```

次の2つの例（"Single Directory with multiple branches - Aggregate DN Resolver"および"Multiple Directories"）は複数のLDAPツリーがサブツリーとして扱えない環境、或いはLDAPサーバが複数の環境向けで、ldap.propertiesとldap-authn-config.xmlを変更します。
LDAPサーバ/LDAPツリーの数に応じて、ldap.properties内の項目を追加してauthn-ldap-config.xmlでそれらを参照するようにします。
また、これらの例は認証処理のための設定なので、SPへ送出する属性をLDAPから取得している場合はattribute-resolver.xmlを変更し、ldap.propertiesに追加した項目を参照するLDAP DataConnectorを追加します。

- LDAPサーバが一台であり複数のLDAPツリーがサブツリーとして扱えない場合の例

/opt/shibboleth-idp/conf/authn/ldap-authn-config.xml の設定例

```
    <bean name="aggregateAuthenticator" class="org.ldaptive.auth.Authenticator">
        <constructor-arg index="0" ref="aggregateDnResolver" />
        <constructor-arg index="1" ref="aggregateAuthHandler" />
    </bean>
    <bean id="aggregateDnResolver" class="org.ldaptive.auth.AggregateDnResolver">
        <constructor-arg index="0" ref="dnResolvers" />
    </bean>
    <bean id="aggregateAuthHandler" class="org.ldaptive.auth.AggregateDnResolver$AuthenticationHandler" p:
authenticationHandlers-ref="authHandlers" />
    <util:map id="dnResolvers">
        <entry key="filter1" value-ref="dnResolver1" />
        <entry key="filter2" value-ref="dnResolver2" />
    </util:map>
    <!-- Define two DN resolvers that use anonymous search against the same directory -->
    <bean id="dnResolver1" class="org.ldaptive.auth.PooledSearchDnResolver" p:baseDn="%{idp.authn.LDAP.baseDN}"
          p:subtreeSearch="%{idp.authn.LDAP.subtreeSearch:false}" p:userFilter="%{idp.authn.LDAP.userFilter}"
          p:connectionFactory-ref="anonSearchPooledConnectionFactory" />
    <bean id="dnResolver2" class="org.ldaptive.auth.PooledSearchDnResolver" p:baseDn="%{idp.authn.LDAP.baseDN2}"
          p:subtreeSearch="%{idp.authn.LDAP.subtreeSearch:false}" p:userFilter="%{idp.authn.LDAP.userFilter2}"
          p:connectionFactory-ref="anonSearchPooledConnectionFactory" />
    <!-- Use the same authentication handler for both DN resolvers -->
    <util:map id="authHandlers">
        <entry key="filter1" value-ref="authHandler" />
        <entry key="filter2" value-ref="authHandler" />
    </util:map> ← </beans>の前に追加

</beans>
```

/opt/shibboleth-idp/conf/ldap.properties の設定例

```
idp.authn.LDAP.authenticator = aggregateAuthenticator ← ldap-authn-config.xmlに追加したclass="org.ldaptive.auth.
Authenticator"のbeanのnameに変更
(省略)
# Search DN resolution, used by anonSearchAuthenticator, bindSearchAuthenticator
# for AD: CN=Users,DC=example,DC=org
idp.authn.LDAP.baseDN = LDAPツリー1のBaseDN ← 変更
idp.authn.LDAP.baseDN2 = LDAPツリー2のBaseDN ← 追加
idp.authn.LDAP.subtreeSearch = true
idp.authn.LDAP.userFilter = (LDAPツリー1の検索キー={user}) ← 変更
idp.authn.LDAP.userFilter2 = (LDAPツリー2の検索キー={user}) ← 追加
(省略)
# LDAP attribute configuration, see attribute-resolver.xml
# Note, this likely won't apply to the use of legacy V2 resolver configurations
idp.attribute.resolver.LDAP.ldapURL = %{idp.authn.LDAP.ldapURL}
idp.attribute.resolver.LDAP.baseDN = %{idp.authn.LDAP.baseDN:undefined}
idp.attribute.resolver.LDAP.baseDN2 = %{idp.authn.LDAP.baseDN2:undefined} ← 追加
idp.attribute.resolver.LDAP.bindDN = %{idp.authn.LDAP.bindDN:undefined}
```

/opt/shibboleth-idp/conf/attribute-resolver.xml の設定例

```
    <resolver:DataConnector id="myLDAP" xsi:type="dc:LDAPDirectory"
        ldapURL="%{idp.attribute.resolver.LDAP.ldapURL}"
        baseDN="%{idp.attribute.resolver.LDAP.baseDN}"
        principal="%{idp.attribute.resolver.LDAP.bindDN}"
        principalCredential="%{idp.attribute.resolver.LDAP.bindDNCredential}"
        noResultIsError="True" ← 追加
        useStartTLS="%{idp.attribute.resolver.LDAP.useStartTLS:true}">
        <resolver:FailoverDataConnector ref="myLDAP2" /> ← 追加(dcより前に追加する必要があります)
        <dc:FilterTemplate>
            <![CDATA[
                %{idp.attribute.resolver.LDAP.searchFilter}
            ]]>
        </dc:FilterTemplate>
        <dc:StartTLSTrustCredential id="LDAPtoIdPCredential" xsi:type="sec:X509ResourceBacked">
            <sec:Certificate>%{idp.attribute.resolver.LDAP.trustCertificates}</sec:Certificate>
        </dc:StartTLSTrustCredential>
    </resolver:DataConnector>
    ↓以下の行を追加
    <resolver:DataConnector id="myLDAP2" xsi:type="dc:LDAPDirectory"
        ldapURL="%{idp.attribute.resolver.LDAP.ldapURL}"
        baseDN="%{idp.attribute.resolver.LDAP.baseDN2}"
        principal="%{idp.attribute.resolver.LDAP.bindDN}"
        principalCredential="%{idp.attribute.resolver.LDAP.bindDNCredential}"
        noResultIsError="True"
        useStartTLS="%{idp.attribute.resolver.LDAP.useStartTLS:true}">
        <dc:FilterTemplate>
            <![CDATA[
                %{idp.attribute.resolver.LDAP.searchFilter}
            ]]>
        </dc:FilterTemplate>
        <dc:StartTLSTrustCredential id="LDAPtoIdPCredential" xsi:type="sec:X509ResourceBacked">
            <sec:Certificate>%{idp.attribute.resolver.LDAP.trustCertificates}</sec:Certificate>
        </dc:StartTLSTrustCredential>
    </resolver:DataConnector>
```

- LDAPサーバが複数である場合の例

/opt/shibboleth-idp/conf/authn/ldap-authn-config.xml の設定例

```
    <alias name="%{idp.authn.LDAP.sslConfig2:certificateTrust2}" alias="sslConfig2" />
    <bean id="certificateTrust2" class="org.ldaptive.ssl.SslConfig">
        <property name="credentialConfig">
            <bean parent="shibboleth.X509ResourceCredentialConfig" p:trustCertificates="%{idp.authn.LDAP.trustCertificates2:
undefined}" />
        </property>
    </bean>
    <bean id="keyStoreTrust2" class="org.ldaptive.ssl.SslConfig">
        <property name="credentialConfig">
            <bean parent="shibboleth.KeystoreResourceCredentialConfig" p:truststore="%{idp.authn.LDAP.trustStore2:undefined}"
/>
        </property>
    </bean>
    <bean name="aggregateAuthenticator" class="org.ldaptive.auth.Authenticator"
          c:resolver-ref="aggregateDnResolver"
          c:handler-ref="aggregateAuthHandler" />

    <!-- Aggregate DN resolution -->
    <bean id="aggregateDnResolver" class="org.ldaptive.auth.AggregateDnResolver"
          c:resolvers-ref="dnResolvers"
          p:allowMultipleDns="true" />
    <util:map id="dnResolvers">
        <entry key="directory1" value-ref="bindSearchDnResolver1" />
        <entry key="directory2" value-ref="bindSearchDnResolver2" />
    </util:map>

    <!-- DN resolver 1 -->
    <bean id="bindSearchDnResolver1" class="org.ldaptive.auth.PooledSearchDnResolver"
          p:baseDn="#{'%{idp.authn.LDAP.baseDN:undefined}'.trim()}"
          p:subtreeSearch="%{idp.authn.LDAP.subtreeSearch:false}"
          p:userFilter="#{'%{idp.authn.LDAP.userFilter:undefined}'.trim()}"
```

```
      p:connectionFactory-ref="bindSearchPooledConnectionFactory" />
<bean id="bindSearchPooledConnectionFactory1" class="org.ldaptive.pool.PooledConnectionFactory"
      p:connectionPool-ref="bindSearchConnectionPool1" />
<bean id="bindSearchConnectionPool1" class="org.ldaptive.pool.BlockingConnectionPool" parent="connectionPool"
      p:connectionFactory-ref="bindSearchConnectionFactory1"
      p:name="search-pool1" />
<bean id="bindSearchConnectionFactory1" class="org.ldaptive.DefaultConnectionFactory"
      p:connectionConfig-ref="bindSearchConnectionConfig1" />
<bean id="bindSearchConnectionConfig1" parent="connectionConfig"
      p:connectionInitializer-ref="bindConnectionInitializer1"
      p:ldapUrl="%{idp.authn.LDAP.ldapURL}" />
<bean id="bindConnectionInitializer1" class="org.ldaptive.BindConnectionInitializer"
      p:bindDn="#{'%{idp.authn.LDAP.bindDN:undefined}'.trim()}">
    <property name="bindCredential">
        <bean class="org.ldaptive.Credential" c:password="%{idp.authn.LDAP.bindDNCredential:undefined}" />
    </property>
</bean>
<!-- DN resolver 2 -->
<bean id="bindSearchDnResolver2" class="org.ldaptive.auth.PooledSearchDnResolver"
      p:baseDn="#{'%{idp.authn.LDAP.baseDN2:undefined}'.trim()}"
      p:subtreeSearch="%{idp.authn.LDAP.subtreeSearch:false}"
      p:userFilter="#{'%{idp.authn.LDAP.userFilter2:undefined}'.trim()}"
      p:connectionFactory-ref="bindSearchPooledConnectionFactory" />
<bean id="bindSearchPooledConnectionFactory2" class="org.ldaptive.pool.PooledConnectionFactory"
      p:connectionPool-ref="bindSearchConnectionPool2" />
<bean id="bindSearchConnectionPool2" class="org.ldaptive.pool.BlockingConnectionPool" parent="connectionPool"
      p:connectionFactory-ref="bindSearchConnectionFactory2"
      p:name="search-pool2" />
<bean id="bindSearchConnectionFactory2" class="org.ldaptive.DefaultConnectionFactory"
      p:connectionConfig-ref="bindSearchConnectionConfig2" />
<bean id="bindSearchConnectionConfig2" parent="connectionConfig"
      p:connectionInitializer-ref="bindConnectionInitializer2"
      p:ldapUrl="%{idp.authn.LDAP.ldapURL2}"
      p:useStartTLS="%{idp.authn.LDAP.useStartTLS2:true}"
      p:useSSL="%{idp.authn.LDAP.useSSL2:false}"
      p:connectTimeout="%{idp.authn.LDAP.connectTimeout2:3000}"
      p:sslConfig-ref="sslConfig2" />
<bean id="bindConnectionInitializer2" class="org.ldaptive.BindConnectionInitializer"
      p:bindDn="#{'%{idp.authn.LDAP.bindDN2:undefined}'.trim()}">
    <property name="bindCredential">
        <bean class="org.ldaptive.Credential" c:password="%{idp.authn.LDAP.bindDNCredential2:undefined}" />
    </property>
</bean>

<!-- Aggregate authentication -->
<bean id="aggregateAuthHandler" class="org.ldaptive.auth.AggregateDnResolver$AuthenticationHandler"
      p:authenticationHandlers-ref="authHandlers" />
<util:map id="authHandlers">
    <entry key="directory1" value-ref="authHandler1" />
    <entry key="directory2" value-ref="authHandler2" />
</util:map>

<!-- Authentication handler 1 -->
<bean id="authHandler1" class="org.ldaptive.auth.PooledBindAuthenticationHandler"
      p:connectionFactory-ref="bindPooledConnectionFactory1" />
<bean id="bindPooledConnectionFactory1" class="org.ldaptive.pool.PooledConnectionFactory"
      p:connectionPool-ref="bindConnectionPool1" />
<bean id="bindConnectionPool1" class="org.ldaptive.pool.BlockingConnectionPool" parent="connectionPool"
      p:connectionFactory-ref="bindConnectionFactory1"
      p:name="bind-pool1" />
<bean id="bindConnectionFactory1" class="org.ldaptive.DefaultConnectionFactory"
      p:connectionConfig-ref="bindConnectionConfig1" />
<bean id="bindConnectionConfig1" parent="connectionConfig"
      p:ldapUrl="%{idp.authn.LDAP.ldapURL}" />
<!-- Authentication handler 2 -->
<bean id="authHandler2" class="org.ldaptive.auth.PooledBindAuthenticationHandler"
      p:connectionFactory-ref="bindPooledConnectionFactory2" />
<bean id="bindPooledConnectionFactory2" class="org.ldaptive.pool.PooledConnectionFactory"
      p:connectionPool-ref="bindConnectionPool2" />
<bean id="bindConnectionPool2" class="org.ldaptive.pool.BlockingConnectionPool" parent="connectionPool"
      p:connectionFactory-ref="bindConnectionFactory2"
      p:name="bind-pool2" />
<bean id="bindConnectionFactory2" class="org.ldaptive.DefaultConnectionFactory"
      p:connectionConfig-ref="bindConnectionConfig2" />
<bean id="bindConnectionConfig2" parent="connectionConfig"
      p:ldapUrl="%{idp.authn.LDAP.ldapURL2}"
```

```
        p:useStartTLS="%{idp.authn.LDAP.useStartTLS2:true}"
        p:useSSL="%{idp.authn.LDAP.useSSL2:false}"
        p:connectTimeout="%{idp.authn.LDAP.connectTimeout2:3000}"
        p:sslConfig-ref="sslConfig2" /> ← </beans>の前に追加

</beans>
```

/opt/shibboleth-idp/conf/ldap.properties の設定例

```
idp.authn.LDAP.authenticator = aggregateAuthenticator ← ldap-authn-config.xmlに追加したclass="org.ldaptive.auth.
Authenticator"のbeanのnameに変更

## Connection properties ##
idp.authn.LDAP.ldapURL                          = LDAPサーバ1のURL ← 変更
idp.authn.LDAP.ldapURL2                          = LDAPサーバ2のURL ← 追加
idp.authn.LDAP.useStartTLS                      = true
idp.authn.LDAP.useStartTLS2                      = true
#idp.authn.LDAP.useSSL                          = false
#idp.authn.LDAP.connectTimeout                  = 3000

## SSL configuration, either jvmTrust, certificateTrust, or keyStoreTrust
idp.authn.LDAP.sslConfig                       = certificateTrust
↓ 追加(certificateTrustかkeyStoreTrustを指定する際はldap-authn-config.xmlに追加したLDAPサーバ2用のclass="org.ldaptive.ssl.
SslConfig"のbeanのidを指定)
idp.authn.LDAP.sslConfig2                        = certificateTrust2 ← 追加
## If using certificateTrust above, set to the trusted certificate's path
idp.authn.LDAP.trustCertificates                = LDAPサーバ1の証明書 ← 変更
idp.authn.LDAP.trustCertificates2                = LDAPサーバ2の証明書 ← 追加
## If using keyStoreTrust above, set to the truststore path
idp.authn.LDAP.trustStore                       = LDAPサーバ1のキーストア ← 変更
idp.authn.LDAP.trustStore2                       = LDAPサーバ2のキーストア ← 追加
(省略)
# Search DN resolution, used by anonSearchAuthenticator, bindSearchAuthenticator
# for AD: CN=Users,DC=example,DC=org
idp.authn.LDAP.baseDN                           = LDAPサーバ1のBaseDN ← 変更
idp.authn.LDAP.baseDN2                           = LDAPサーバ2のBaseDN ← 追加
idp.authn.LDAP.subtreeSearch                   = true
idp.authn.LDAP.userFilter                       = (LDAPサーバ1の検索キー={user}) ← 変更
idp.authn.LDAP.userFilter2                       = (LDAPサーバ2の検索キー={user}) ← 追加
# bind search configuration
# for AD: idp.authn.LDAP.bindDN=adminuser@domain.com
idp.authn.LDAP.bindDN                           = LDAPサーバ1のBindDN ← 変更
idp.authn.LDAP.bindDN2                           = LDAPサーバ2のBindDN ← 追加
idp.authn.LDAP.bindDNCredential                 = LDAPサーバ1のBindDNパスワード ← 変更
idp.authn.LDAP.bindDNCredential2                 = LDAPサーバ2のBindDNパスワード ← 追加
(省略)
# LDAP attribute configuration, see attribute-resolver.xml
# Note, this likely won't apply to the use of legacy V2 resolver configurations
idp.attribute.resolver.LDAP.ldapURL             = %{idp.authn.LDAP.ldapURL}
idp.attribute.resolver.LDAP.ldapURL2             = %{idp.authn.LDAP.ldapURL2} ← 追加
idp.attribute.resolver.LDAP.baseDN              = %{idp.authn.LDAP.baseDN:undefined}
idp.attribute.resolver.LDAP.baseDN2              = %{idp.authn.LDAP.baseDN2:undefined} ← 追加
idp.attribute.resolver.LDAP.bindDN              = %{idp.authn.LDAP.bindDN:undefined}
idp.attribute.resolver.LDAP.bindDN2              = %{idp.authn.LDAP.bindDN2:undefined} ← 追加
idp.attribute.resolver.LDAP.bindDNCredential    = %{idp.authn.LDAP.bindDNCredential:undefined}
idp.attribute.resolver.LDAP.bindDNCredential2    = %{idp.authn.LDAP.bindDNCredential2:undefined} ← 追加
idp.attribute.resolver.LDAP.useStartTLS         = %{idp.authn.LDAP.useStartTLS:true}
idp.attribute.resolver.LDAP.useStartTLS2         = %{idp.authn.LDAP.useStartTLS2:true} ← 追加
idp.attribute.resolver.LDAP.trustCertificates   = %{idp.authn.LDAP.trustCertificates:undefined}
idp.attribute.resolver.LDAP.trustCertificates2   = %{idp.authn.LDAP.trustCertificates2:undefined} ← 追加
idp.attribute.resolver.LDAP.searchFilter        = (LDAPサーバ1の検索キー=$resolutionContext.principal) ← 変更
idp.attribute.resolver.LDAP.searchFilter2        = (LDAPサーバ2の検索キー=$resolutionContext.principal) ← 追加
idp.attribute.resolver.LDAP.returnAttributes    = cn,homephone,mail
```

/opt/shibboleth-idp/conf/attribute-resolver.xml の設定例

```
    <resolver:DataConnector id="myLDAP" xsi:type="dc:LDAPDirectory"
        ldapURL="%{idp.attribute.resolver.LDAP.ldapURL}"
        baseDN="%{idp.attribute.resolver.LDAP.baseDN}"
        principal="%{idp.attribute.resolver.LDAP.bindDN}"
        principalCredential="%{idp.attribute.resolver.LDAP.bindDNCredential}"
        noResultIsError="True" ← 追加
        useStartTLS="%{idp.attribute.resolver.LDAP.useStartTLS:true}">
        <resolver:FailoverDataConnector ref="myLDAP2" /> ← 追加(dcより前に追加する必要があります)
        <dc:FilterTemplate>
            <![CDATA[
                %{idp.attribute.resolver.LDAP.searchFilter}
            ]]>
        </dc:FilterTemplate>
        <dc:StartTLSTrustCredential id="LDAPtoIdPCredential" xsi:type="sec:X509ResourceBacked">
            <sec:Certificate>%{idp.attribute.resolver.LDAP.trustCertificates}</sec:Certificate>
        </dc:StartTLSTrustCredential>
    </resolver:DataConnector>
    ↓以下の行を追加
    <resolver:DataConnector id="myLDAP2" xsi:type="dc:LDAPDirectory"
        ldapURL="%{idp.attribute.resolver.LDAP.ldapURL2}"
        baseDN="%{idp.attribute.resolver.LDAP.baseDN2}"
        principal="%{idp.attribute.resolver.LDAP.bindDN2}"
        principalCredential="%{idp.attribute.resolver.LDAP.bindDNCredential2}"
        noResultIsError="True"
        useStartTLS="%{idp.attribute.resolver.LDAP.useStartTLS2:true}">
        <dc:FilterTemplate>
            <![CDATA[
                %{idp.attribute.resolver.LDAP.searchFilter2}
            ]]>
        </dc:FilterTemplate>
        <dc:StartTLSTrustCredential id="LDAPtoIdPCredential" xsi:type="sec:X509ResourceBacked">
            <sec:Certificate>%{idp.attribute.resolver.LDAP.trustCertificates2}</sec:Certificate>
        </dc:StartTLSTrustCredential>
    </resolver:DataConnector>
```

attribute-filter.xmlの変更はDataConnector "myLDAP" にデータが見つからない場合エラーとして扱うようにし、フェイルオーバー処理としてDataConnector "myLDAP2" から改めてデータを検索する設定になります。

> ⚠ 1つ目の方法と同様、同一のIDが複数のLDAPツリー上に存在すると問題になりますので、uidがこの条件を満たさない場合は他のLDAP属性をID(principal)として使うようにしてください。

Shibboleth IdP 3の高度な認証設定

個別のページに移動

> ⛔ 以下でご紹介しているExtendedフローはdeprecatedとなりShibboleth IdPバージョン5で削除される予定です。現行バージョンで当該機能を使っている方はMFAへの移行をご検討ください。

> ⚠ 以下で4.1でもまだ有効な簡便な方法を説明していますが、3.3以降であればより汎用的で複雑な挙動が実現できるMFAによる方法もご参照ください。

- 認証フローの階層化
- Password認証フローのExtendedフロー
- 参考

Shibboleth IdP 3以降の高度な認証設定についてのドキュメントです。4.1以降で書式が大幅に変更になりましたのでそれに特化した記述になっております。本ドキュメントはSAML 2.0で認証の切り替えを行うことを目的としており、SAML1は対象外です（LevelXを用いた認証要求はできません）。

> ⚠ SAML1を使うことにより本設定の制約を迂回できることを避けるため、SPにおいてはshibboleth2.xmlにてSAML1の機能を無効化することをお勧めします。

本ドキュメントで使用する認証コンテクストと認証フローの関係を下記に示します。

⚠ ここでRemoteUserはパスワード認証とクライアント証明書認証の中間の強度の認証の例として挙げております。実際に運用する場合は別途用意した認証フローで置き換えてください。また、レベルとの対応付けも本ドキュメント独自のもので、例示として使用しています。

認証コンテクスト	略称	認証フロー
urn:mace:gakunin.jp:idprivacy:ac:classes:Level1	Level1	Password
urn:mace:gakunin.jp:idprivacy:ac:classes:Level2	Level2	RemoteUser
urn:mace:gakunin.jp:idprivacy:ac:classes:Level3	Level3	X509

この認証コンテクストとは別に、PasswordやX509は固有の認証コンテクストを持っていますが、ここでは使用しません。

挙動の説明で使用するSPについて下記に示します。

SP	要求する認証
SP_a	なし
SP_b	urn:mace:gakunin.jp:idprivacy:ac:classes:Level1
SP_c	urn:mace:gakunin.jp:idprivacy:ac:classes:Level2
SP_d	urn:mace:gakunin.jp:idprivacy:ac:classes:Level3

認証フローの階層化

設定

1. 既に認証済みの認証フローを優先するために、conf/authn/authn.propertiesのidp.authn.favorSSOをアンコメントしtrueに設定します。

conf/authn/authn.properties

```
 # Whether to prioritize "active" results when an SP requests more than
 # one possible matching login method (V2 behavior was to favor them)
-#idp.authn.favorSSO = false
+idp.authn.favorSSO = true
```

2. 各認証フローのsupportedPrincipalsプロパティに下記を追加します。

認証フロー	supportedPrincipalsプロパティ
Password	継承元のshibboleth.AuthenticationFlowで定義されているsupportedPrincipals, Level1
RemoteUser	Level2, Level1
X509	Level3, Level2, Level1

conf/authn/authn.properties

```
@@ -56,10 +56,11 @@
 # Unset if using customized Principals per validator
 #idp.authn.Password.addDefaultPrincipals = true
 # The Principal collection below is the typical default if not otherwise noted.
-#idp.authn.Password.supportedPrincipals = ¥
-#    saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport, ¥
-#    saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:Password, ¥
-#    saml1/urn:oasis:names:tc:SAML:1.0:am:password
+idp.authn.Password.supportedPrincipals = ¥
+    saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport, ¥
+    saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:Password, ¥
+    saml1/urn:oasis:names:tc:SAML:1.0:am:password, ¥
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level1
 # Validators are controlled in password-authn-config.xml

 #### Password Backends ####
@@ -97,7 +98,9 @@
 idp.authn.RemoteUser.supportedPrincipals = ¥
  saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:X509, ¥
  saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient, ¥
- saml1/urn:ietf:rfc:2246
+ saml1/urn:ietf:rfc:2246, ¥
+ saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level2, ¥
+ saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level1

 #### RemoteUserInternal ####

@@ -137,7 +140,10 @@
 idp.authn.X509.supportedPrincipals = ¥
     saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:X509, ¥
     saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient, ¥
-    saml1/urn:ietf:rfc:2246
+    saml1/urn:ietf:rfc:2246, ¥
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level3, ¥
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level2, ¥
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level1

 #### X509Internal ####
```

3. conf/relying-party.xmlのshibboleth.DefaultRelyingParty内のSAML2.SSOにdefaultAuthenticationMethodsプロパティを設定します。

conf/relying-party.xml

```
    <bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">
        <property name="profileConfigurations">
            <list>
                <bean parent="Shibboleth.SSO" p:postAuthenticationFlows="attribute-release" />
                <ref bean="SAML1.AttributeQuery" />
                <ref bean="SAML1.ArtifactResolution" />
-               <bean parent="SAML2.SSO" p:postAuthenticationFlows="attribute-release" />
+               <bean parent="SAML2.SSO" p:postAuthenticationFlows="attribute-release">
+                   <property name="defaultAuthenticationMethods">
+                       <list>
+                           <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                               c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                       </list>
+                   </property>
+               </bean>
                <ref bean="SAML2.ECP" />
                <ref bean="SAML2.Logout" />
                <ref bean="SAML2.AttributeQuery" />
                <ref bean="SAML2.ArtifactResolution" />
                <ref bean="Liberty.SSOS" />
            </list>
        </property>
    </bean>
```

必要であればRelyingPartyOverridesのほうのSAML2.SSOにも同様の設定を追加してください。

4. この設定で問題がある場合、つまり許容されているレベルでもより高レベルの認証が求められる場合は、authn.propertiesの idp.authn.*.order を調整してください。より弱い認証のorderを小さい数にすれば、より高い優先度で表示されるようになります。

挙動

未認証の場合

1. SP_aもしくはSP_bからIdPにリダイレクトされると、Level1のPassword認証フローのログインページが表示されます。
2. SP_cからIdPにリダイレクトされると、Level2のRemoteUser認証フローのためのログインページやダイアログが表示されます。
3. SP_dからIdPにリダイレクトされると、Level3のX509認証フローのログインページが表示されます。

Level1が認証済みの場合

1. SP_aもしくはSP_bからIdPにリダイレクトされると、Level1のPassword認証フローが認証済みのためユーザ同意画面もしくは認証後のSPの画面が表示されます。
2. SP_cからIdPにリダイレクトされると、Level2のRemoteUser認証フローのためのログインページやダイアログが表示されます。
3. SP_dからIdPにリダイレクトされると、Level3のX509認証フローのログインページが表示されます。

Level2が認証済みの場合

1. SP_aもしくはSP_bからIdPにリダイレクトされると、Level2のRemoteUser認証フローが認証済みのためユーザ同意画面もしくは認証後のSPの画面が表示されます。
2. SP_cからIdPにリダイレクトされると、Level2のRemoteUser認証フローが認証済みのためユーザ同意画面もしくはSPの画面が表示されます。
3. SP_dからIdPにリダイレクトされると、Level3のX509認証フローのログインページが表示されます。

Level3が認証済みの場合

1. SP_aもしくはSP_bからIdPにリダイレクトされると、Level3のX509認証フローが認証済みのためユーザ同意画面もしくは認証後のSPの画面が表示されます。
2. SP_cからIdPにリダイレクトされると、Level3のX509認証フローが認証済みのためユーザ同意画面もしくは認証後のSPの画面が表示されます。
3. SP_dからIdPにリダイレクトされると、Level3のX509認証フローが認証済みのためユーザ同意画面もしくは認証後のSPの画面が表示されます。

Password認証フローのExtendedフロー

設定

1. 認証フローの階層化の設定を行ってください。
2. conf/authn/authn.properties のauthn/PasswordにExtendedフローで利用するLevel2, Level3を追加します。

conf/authn/authn.properties

```
     saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport, ¥
     saml2/urn:oasis:names:tc:SAML:2.0:ac:classes:Password, ¥
     saml1/urn:oasis:names:tc:SAML:1.0:am:password, ¥
-    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level1
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level1, ¥
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level2, ¥
+    saml2/urn:mace:gakunin.jp:idprivacy:ac:classes:Level3
 # Validators are controlled in password-authn-config.xml
```

3. conf/authn/password-authn-config.xmlでExtendedフローのbeanをアンコメントし、下記の設定を行います。もし当該ファイルにこの部分が存在しなければ、最終行の1つ上にこの部分を挿入してください。
    - shibboleth.authn.Password.ExtendedFlowsのc:_0に、ExtendedフローとするRemoteUserとX509を設定します。
    - shibboleth.authn.Password.PrincipalOverrideに、Password認証フローで認証するLevel1を追加します。Level2やLevel3を除いているところがポイントです。

conf/authn/password-authn-config.xml

```
     <!--
     Configuration of "extended" login methods to offer in the password login form.
     The String bean is a regular expression identifying the flows to offer. These flows
     must also be enabled at the "top" level to be available for use.
     The ExtendedFlowParameters bean can be used to transfer custom parameters from the
     login form into the context tree for use later by other flows.
     The last bean provides the set of custom Principals to use for results produced by the
     Password flow itself. You would use this if you need the Password flow to run as a shell
     to run the "extended" login methods, but want to limit its own results more narrowly.
     -->
-    <!--
-    <bean id="shibboleth.authn.Password.ExtendedFlows" class="java.lang.String" c:_0="" />
+    <bean id="shibboleth.authn.Password.ExtendedFlows" class="java.lang.String" c:_0="RemoteUser|X509" />
     <util:list id="shibboleth.authn.Password.ExtendedFlowParameters">
     </util:list>
     <util:list id="shibboleth.authn.Password.PrincipalOverride">
         <bean parent="shibboleth.SAML2AuthnContextClassRef"
             c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport" />
         <bean parent="shibboleth.SAML2AuthnContextClassRef"
             c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:Password" />
         <bean parent="shibboleth.SAML1AuthenticationMethod"
             c:method="urn:oasis:names:tc:SAML:1.0:am:password" />
+        <!-- -->
+        <bean parent="shibboleth.SAML2AuthnContextClassRef"
+            c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
     </util:list>
-    -->
```

4. Shibboleth IdP 4.1および4.2にはバグがありますので手動でaddDefaultPrincipalsをfalseにしてください。

conf/authn/authn.properties

```
 # Unset if using customized Principals per validator
-#idp.authn.Password.addDefaultPrincipals = true
+idp.authn.Password.addDefaultPrincipals = false
 # The Principal collection below is the typical default if not otherwise noted.
```

5. Shibboleth IdP 4.0.0および4.0.1をお使いの場合および以降のバージョンでも以前のバージョンからアップデートしている場合は、login.vmにバグがあり追加のボタンが表示されませんので、以下の修正を行ってください。

views/login.vm

```
            #end

            #foreach ($extFlow in $extendedAuthenticationFlows)
-             #if ($authenticationContext.isAcceptable($extFlow) and $extFlow.apply(profileRequestContext))
+             #if ($authenticationContext.isAcceptable($extFlow) and $extFlow.test(profileRequestContext))
                <div class="form-element-wrapper">
                  <button class="form-element form-button" type="submit" name="_eventId_$extFlow.getId()">
                    #springMessageText("idp.login.$extFlow.getId().replace('authn/','')", $extFlow.getId().replace('authn
/',''))
```

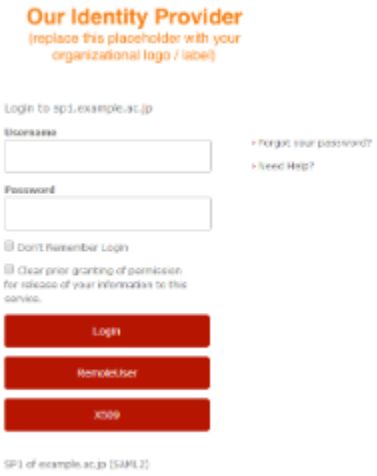
挙動

各SPからIdPにリダイレクトされた時に表示されるPassword認証フローのログインページを下記に示します。

1. SP_aおよびSP_bからの場合
   Password認証フローのための「Username」と「Passowrd」のフォームと「Login」ボタン、およびExtendedフローであるRemoteUser認証フローとX509認証フローのためのボタン「RemoteUser」と「X509」が表示されます。

   「Login」とボタンの形状が同じで紛らわしいですが、Extendedフローを利用する場合は上部のフォーム（「Username」と「Passowrd」）の入力は不要です。

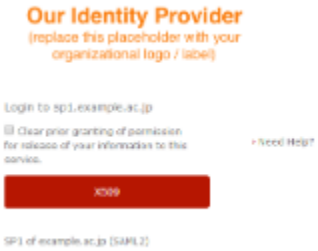
図1. SPaおよびSPbからの場合



2. SP_cからの場合
   ExtendedフローであるRemoteUser認証フローとX509認証フローのためのボタン「RemoteUser」と「X509」が表示されます。Level1のPassword認証フローは表示されません。

図2. SPcからの場合

3. SP_dからの場合
   Extendedフローのうち`Level3`以上であるX509認証フローのためのボタン「X509」が表示されます。`Level1`のPassword認証フロー、および`Level2`のX509認証フローは表示されません。
   図3. SPdからの場合



参考

高度な認証フローを設定する上で、参考になるドキュメントを下記に示します。

- [Shibboleth wiki] AuthenticationConfiguration
- [Shibboleth wiki] AuthenticationFlowSelection
- [Shibboleth wiki] PasswordAuthnConfiguration|ExtendedFlows
  の"Extended Flows"
- [Shibboleth wiki] Orchestrating Multiple Authentication Methods and Contexts - The Multi-Context Broker (MCB)
- [Shibboleth wiki] Configuring the IdP for the Multi-Context Broker Model
- [Shibboleth wiki] Replicating Multi-Context Broker Functionality (Duo + Username/Password with user-opt-in forcing Duo)
- [Shibboleth wiki] SP-driven Duo opt-in
  （リンク先にsystem/以下のファイルを編集している箇所がありますが推奨されていません）
- 3.3向け
  [Shibboleth wiki] MultiFactorAuthnConfiguration

MultiFactor認証フロー(MFA)を用いた認証設定

個別のページに移動

- 共通設定 (General Configuration)
- 直接的なフロー選択 (Directly Selecting Flows)
- プログラムでのフロー選択 (Programmatically Selecting Flows)
- 遷移の完全なコントロール (Full Control Over Transitions)
- 参考

Shibboleth IdP 3.3より導入されたMultiFactor認証フロー(MFA)の認証設定についてのドキュメントです。本ドキュメントはSAML 2.0で認証の切り替えを行うことを目的としており、SAML1は対象外です（LevelXを用いた認証要求はできません）。

> ⚠ SAML1を使うことにより本設定の制約を迂回できることを避けるため、SPにおいてはshibboleth2.xmlにてSAML1の機能を無効化することをお勧めします。

MultiFactor認証フローは、シンプルないし複雑な認証シーケンスを作るために複数の認証フローを組み合わせるスクリプト記述可能な方法を提供します。

共通設定 (General Configuration)

MultiFactor認証フローの設定は、`conf/authn/mfa-authn-config.xml`で行います。

また、`conf/idp.properties`の`idp.authn.flows`でMultiFactor認証フローを有効にします。注意すべき点として、MultiFactor認証フローからルールやスクリプトを介して呼び出される認証フローについては、意図していない方法で当該認証フローが実行されるかもしれないため、`idp.authn.flows`では有効にすべきではないとされています。

conf/idp.properties

```
 # Regular expression matching login flows to enable, e.g. IPAddress|Password
-idp.authn.flows= Password
+idp.authn.flows= MFA
```

直接的なフロー選択 (Directly Selecting Flows)

もっと簡単なルールは、最初の認証フローが成功した場合に、次に実行する認証フローを指定します。

下記の例は、最初にPassword認証フローによる認証を行い、Password認証フローの認証が成功した場合にX509認証フローの認証が行われます。X509認証フローの認証が成功すると認証成功となります。

conf/authn/mfa-authn-config.xml

```
<util:map id="shibboleth.authn.MFA.TransitionMap">
    <!-- Run authn/Password first. -->
    <entry key="">
        <bean parent="shibboleth.authn.MFA.Transition" p:nextFlow="authn/Password" />
    </entry>

    <!-- If that returns "proceed", run authn/X509 next. -->
    <entry key="authn/Password">
        <bean parent="shibboleth.authn.MFA.Transition" p:nextFlow="authn/X509" />
    </entry>

    <!-- An implicit final rule will return whatever the second flow returns. -->
</util:map>
```

プログラムでのフロー選択 (Programmatically Selecting Flows)

より複雑なルールを実現するには、Script、Spring Expression、もしくはJavaで記述した関数を実行します。

下記の例は、以下の認証シーケンスを実現しています。

1. 最初にPassword認証フローを実行します。
2. ステップ1のPassword認証フローが認証成功し認証要求を満たすのに十分であればステップ3を、そうでなければステップ5に遷移します。
3. ステップ1によって識別されたユーザの属性 `allowedLoginMethods`を取得します。
4. 属性 `allowedLoginMethods`が存在し、かつ属性値に**Password**が含まれていれば、Password認証フローのみで認証成功になります。そうでなければステップ5に遷移します。
5. X509認証フローを実行します。
6. X509認証フローが認証成功すれば、Password認証フローとX509認証フローの認証結果が一つにマージされます。
7. 両方の認証フローが認証成功であれば認証成功となり、そうでなければ認証失敗となります。

なお、`conf/attribute-resolver.xml`に属性`allowedLoginMethods`を追加する必要があります。

conf/authn/mfa-authn-config.xml

```
<util:map id="shibboleth.authn.MFA.TransitionMap">
    <!-- Run authn/Password first. -->
    <entry key="">
        <bean parent="shibboleth.authn.MFA.Transition" p:nextFlow="authn/Password" />
    </entry>

    <!--
    Second rule runs a function if authn/Password succeeds, to determine whether an additional
    factor is required.
    -->
    <entry key="authn/Password">
        <bean parent="shibboleth.authn.MFA.Transition" p:nextFlowStrategy-ref="checkSecondFactor" />
    </entry>

    <!-- An implicit final rule will return whatever the second flow returns. -->
</util:map>

<!-- Example script to see if second factor is required. -->
<bean id="checkSecondFactor" parent="shibboleth.ContextFunctions.Scripted" factory-method="inlineScript"
        p:customObject-ref="shibboleth.AttributeResolverService">
    <constructor-arg>
        <value>
        <![CDATA[
            nextFlow = "authn/X509";

            // Go straight to second factor if we have to, or set up for an attribute lookup first.
            authCtx = input.getSubcontext("net.shibboleth.idp.authn.context.AuthenticationContext");
            mfaCtx = authCtx.getSubcontext("net.shibboleth.idp.authn.context.MultiFactorAuthenticationContext");
            if (mfaCtx.isAcceptable()) {
                // Attribute check is required to decide if first factor alone is enough.
                resCtx = input.getSubcontext(
                    "net.shibboleth.idp.attribute.resolver.context.AttributeResolutionContext", true);

                // Look up the username
                usernameLookupStrategyClass = Java.type("net.shibboleth.idp.session.context.navigate.
CanonicalUsernameLookupStrategy");
                usernameLookupStrategy = new usernameLookupStrategyClass();
                resCtx.setPrincipal(usernameLookupStrategy.apply(input));


                // resolve the attribute to determine if a first factor is sufficient
                resCtx.getRequestedIdPAttributeNames().add("allowedLoginMethods");
                resCtx.resolveAttributes(custom);

                // Check for an attribute value that authorizes use of first factor.
                attribute = resCtx.getResolvedIdPAttributes().get("allowedLoginMethods");
                valueType =  Java.type("net.shibboleth.idp.attribute.StringAttributeValue");
                if (attribute != null && attribute.getValues().contains(new valueType("Password"))) {
                    nextFlow = null;
                }

                input.removeSubcontext(resCtx);   // cleanup
            }

            nextFlow;   // pass control to second factor or end with the first
        ]]>
    </value>
    </constructor-arg>
</bean>
```

遷移の完全なコントロール (Full Control Over Transitions)

最も複雑なルールを実現するために、Spring WebFlowイベントに基づいて完全に遷移を制御できます。

下記の例ではShibboleth IdPバージョン2向けにNIIと金沢大学で共同開発したGUARDプラグインと同様な認証シーケンスを実現します。GUARDプラグインについては、2015年2月に金沢大学の松平様が発表された『大学統合認証基盤における多要素認証について』の12ページをご参照ください。下記の例での認証コンテクストと認証フローの関係を以下に示します。

認証コンテクスト	略称	認証フロー
urn:mace:gakunin.jp:idprivacy:ac:classes:Level1	Level1	パスワード
urn:mace:gakunin.jp:idprivacy:ac:classes:Level2	Level2	(機関内) パスワード **OR** RemoteUser **OR** X509 (機関外) RemoteUser **OR** X509
urn:mace:gakunin.jp:idprivacy:ac:classes:Level3	Level3	RemoteUser **AND** X509

制限事項

- Level1のパスワード認証後に、Level2のパスワード認証はSSOされません。
- Level2のRemoteUser認証後に、Level3のRemoteUser認証はSSOされません。(ただし、BASIC認証に関して言えばブラウザが自動的にユーザー名とパスワードを送信するためエンドユーザーがユーザー名やパスワードを再度入力する必要はありません)

設定

1. conf/authn/mfa-authn-config.xmlで各Levelに応じた認証設定を行います。

conf/authn/mfa-authn-config.xml

```
<util:map id="shibboleth.authn.MFA.TransitionMap">
    <!-- First rule calls a flow to display a view to select a method to run. -->
    <entry key="">
        <bean parent="shibboleth.authn.MFA.Transition" p:nextFlow="custom/methodChooser" />
    </entry>

    <!-- Second rule decides what to call based on event signaled by the view. -->
    <entry key="custom/methodChooser">
        <bean parent="shibboleth.authn.MFA.Transition">
            <property name="nextFlowStrategyMap">
                <map>
                    <!-- Maps event to a flow -->
                    <!-- Level1 -->
                    <entry key="ChooseLevel1" value="authn/Level1" />

                    <!-- Level2 -->
                    <entry key="ChoosePassword"   value="authn/Password" />
                    <entry key="ChooseRemoteUser" value="authn/RemoteUser" />
                    <entry key="ChooseX509"       value="authn/X509" />


                    <!-- Level3 -->
                    <entry key="ChooseLevel3" value="authn/RemoteUser4Level3" />
                </map>
            </property>
        </bean>
    </entry>

    <!-- Level3 -->
    <entry key="authn/RemoteUser4Level3">
        <bean parent="shibboleth.authn.MFA.Transition" p:nextFlow="authn/X509" />
    </entry>

    <!-- An implicit final rule will return whatever the final flow returns. -->
</util:map>
```

   Level2, Level3で異なる認証フローを使用したい場合は16行目から23行目、および31行目の<entry>を変更します。
2. 下記ファイルをダウンロードして配置します。

ファイル名	配置先ディレクトリ
methodChooser-flow.xml	flow/custom/methodChooser/

chooser.vm	views/

Level2, Level3で異なる認証フローを使用したい場合は、methodChooser-flow.xmlの30行目の Password|RemoteUser|X509 の部分と、37行目から40行目の<transition>、および47行目から49行目の<end-state>を変更します。

methodChooser-flow.xml

```
    <view-state id="DisplayChooserWebViewForLevel2" view="chooser">
        <on-render>
            <evaluate expression="environment" result="viewScope.environment" />
            <evaluate expression="opensamlProfileRequestContext" result="viewScope.profileRequestContext" />
            <evaluate expression="opensamlProfileRequestContext.getSubcontext(T(net.shibboleth.idp.authn.context.
AuthenticationContext))" result="viewScope.authenticationContext" />
            <evaluate expression="authenticationContext.getAvailableFlows().values().?[id matches 'authn/
(Password|RemoteUser|X509)']" result="viewScope.availableAuthenticationFlows" />
            <evaluate expression="authenticationContext.getSubcontext(T(net.shibboleth.idp.ui.context.
RelyingPartyUIContext))" result="viewScope.rpUIContext" />
            <evaluate expression="T(net.shibboleth.utilities.java.support.codec.HTMLEncoder)" result="viewScope.encoder" />
            <evaluate expression="flowRequestContext.getExternalContext().getNativeRequest()" result="viewScope.request" />
            <evaluate expression="flowRequestContext.getExternalContext().getNativeResponse()" result="viewScope.response" />
            <evaluate expression="flowRequestContext.getActiveFlow().getApplicationContext().containsBean('shibboleth.
CustomViewContext') ? flowRequestContext.getActiveFlow().getApplicationContext().getBean('shibboleth.CustomViewContext') :
null" result="viewScope.custom" />
        </on-render>

        <transition on="ChoosePassword"   to="ChoosePassword" />
        <transition on="ChooseRemoteUser" to="ChooseRemoteUser" />
        <transition on="ChooseX509"       to="ChooseX509" />
    </view-state>

    <!-- Level1 -->
    <end-state id="ChooseLevel1" />

    <!-- Level2 -->
    <end-state id="ChoosePassword" />
    <end-state id="ChooseRemoteUser" />
    <end-state id="ChooseX509" />

    <!-- Level3 -->
    <end-state id="ChooseLevel3" />
```

3. Level1のパスワード認証を用意します。

```
# mkdir -p flows/authn/Level1
# cp system/flows/authn/password-authn-flow.xml Level1-flow.xml
# cp system/flows/authn/password-authn-beans.xml Level1-beans.xml
# sed -i 's/password-authn-beans.xml/Level1-beans.xml/' Level1-flow.xml
```

4. Level3のRemoteUser認証を用意します。

```
# mkdir -p flows/authn/RemoteUser4Level3
# cp system/flows/authn/remoteuser-authn-flow.xml flows/authn/RemoteUser4Level3/RemoteUser4Level3-flow.xml
# cp system/flows/authn/remoteuser-authn-beans.xml flows/authn/RemoteUser4Level3/RemoteUser4Level3-beans.xml
# sed -i 's/remoteuser-authn-beans.xml/RemoteUser4Level3-beans.xml/' flows/authn/RemoteUser4Level3/RemoteUser4Level3-flow.xml
# sed -i 's/remoteuser-authn-config.xml/RemoteUser4Level3-config.xml/' flows/authn/RemoteUser4Level3/RemoteUser4Level3-beans.
xml

# cp conf/authn/remoteuser-authn-config.xml conf/authn/RemoteUser4Level3-config.xml
# sed -i 's,Authn/RemoteUser,Authn/RemoteUser4Level3,' conf/authn/remoteuser-authn-config.xml

# cp webapp/WEB-INF/web.xml edit-webapp/WEB-INF/web.xml
```

edit-webapp/WEB-INF/web.xml

```
 @@ -113,4 +113,15 @@
     </servlet-mapping>

+    <!-- Servlet protected by container used for RemoteUser authentication (Level3) -->
+    <servlet>
+        <servlet-name>RemoteUser4Level3AuthHandler</servlet-name>
+        <servlet-class>net.shibboleth.idp.authn.impl.RemoteUserAuthServlet</servlet-class>
+        <load-on-startup>2</load-on-startup>
+    </servlet>
+    <servlet-mapping>
+        <servlet-name>RemoteUser4Level3AuthHandler</servlet-name>
+        <url-pattern>/Authn/RemoteUser4Level3</url-pattern>
+    </servlet-mapping>
+
     <!-- Servlet protected by container used for X.509 authentication -->
     <servlet>
```

```
# bin/build.sh
```

5. Shibboleth IdP 3の高度な認証設定に従い、conf/authn/general-authn.xmlの設定をします。

conf/authn/general-authn.xml

```
@@ -54,21 +54,89 @@
          <bean id="authn/External" parent="shibboleth.AuthenticationFlow"
             p:nonBrowserSupported="false" />

+        <bean id="authn/Level1" parent="shibboleth.AuthenticationFlow">
+            <property name="supportedPrincipals">
+                <list>
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                </list>
+            </property>
+        </bean>
+
+        <bean id="authn/Level2" parent="shibboleth.AuthenticationFlow">
+            <property name="supportedPrincipals">
+                <list>
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                </list>
+            </property>
+        </bean>
+
+        <bean id="authn/Level3" parent="shibboleth.AuthenticationFlow">
+            <property name="supportedPrincipals">
+                <list>
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level3" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                </list>
+            </property>
+        </bean>
+
+        <bean id="authn/Password" parent="shibboleth.AuthenticationFlow"
+                p:passiveAuthenticationSupported="true"
+                p:forcedAuthenticationSupported="true">
+            <property name="supportedPrincipals">
```

```
+                <list>
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:Password" />
+                    <bean parent="shibboleth.SAML1AuthenticationMethod"
+                        c:method="urn:oasis:names:tc:SAML:1.0:am:password" />
+                    <!-- GUARD -->
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                </list>
+            </property>
+        </bean>
+
         <bean id="authn/RemoteUser" parent="shibboleth.AuthenticationFlow"
-            p:nonBrowserSupported="false" />
+            p:nonBrowserSupported="false">
+            <property name="supportedPrincipals">
+                <list>
+                    <!-- GUARD -->
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                </list>
+            </property>
+        </bean>

         <bean id="authn/RemoteUserInternal" parent="shibboleth.AuthenticationFlow" />

         <bean id="authn/X509" parent="shibboleth.AuthenticationFlow"
                 p:nonBrowserSupported="false">
             <property name="supportedPrincipals">
                 <list>
                     <bean parent="shibboleth.SAML2AuthnContextClassRef"
                         c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:X509" />
                     <bean parent="shibboleth.SAML2AuthnContextClassRef"
                         c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient" />
                     <bean parent="shibboleth.SAML1AuthenticationMethod"
                         c:method="urn:ietf:rfc:2246" />
+                    <!-- GUARD -->
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
                 </list>
             </property>
         </bean>

@@ -89,3 +157,14 @@
-        <bean id="authn/Password" parent="shibboleth.AuthenticationFlow"
-                p:passiveAuthenticationSupported="true"
-                p:forcedAuthenticationSupported="true" />
+        <bean id="authn/RemoteUser4Level3" parent="shibboleth.AuthenticationFlow"
+                p:nonBrowserSupported="false">
+            <property name="supportedPrincipals">
+                <list>
+                    <!-- GUARD -->
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level3" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                        c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                </list>
+            </property>
+        </bean>
@@ -112,22 +191,29 @@
         <bean id="authn/MFA" parent="shibboleth.AuthenticationFlow"
```

```
                p:passiveAuthenticationSupported="true"
                p:forcedAuthenticationSupported="true">
            <!--
            The list below almost certainly requires changes, and should generally be the
            union of any of the separate factors you combine in your particular MFA flow
            rules. The example corresponds to the example in mfa-authn-config.xml that
            combines IPAddress with Password.
            -->
            <property name="supportedPrincipals">
                <list>
                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
                        c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocol" />
                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
                        c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport" />
                    <bean parent="shibboleth.SAML2AuthnContextClassRef"
                        c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:Password" />
                    <bean parent="shibboleth.SAML1AuthenticationMethod"
                        c:method="urn:oasis:names:tc:SAML:1.0:am:password" />
+                   <!-- GUARD -->
+                   <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                       c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level1" />
+                   <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                       c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level2" />
+                   <bean parent="shibboleth.SAML2AuthnContextClassRef"
+                       c:classRef="urn:mace:gakunin.jp:idprivacy:ac:classes:Level3" />
                </list>
            </property>
        </bean>
```

6. Level2のPassword認証フローに機関のIPアドレスレンジ(下記例では203.0.113.0/24)を設定します。

conf/authn/general-authn.xml

```
@@ -90,3 +90,4 @@
        <bean id="authn/Password" parent="shibboleth.AuthenticationFlow"
+               p:activationCondition-ref="authn.PasswordActivationCondition"
                p:passiveAuthenticationSupported="true"
                p:forcedAuthenticationSupported="true">
@@ -241,1 +242,9 @@
+
+    <!--
+    Activation Condition
+    -->
+    <bean id="authn.PasswordActivationCondition" class="org.opensaml.profile.logic.IPRangePredicate"
+            p:httpServletRequest-ref="shibboleth.HttpServletRequest"
+            p:ranges="#{ '203.0.113.0/24' }" >
+    </bean>
 </beans>
```

参考

- [Shibboleth wiki] MultiFactorAuthnConfiguration