2008年度実証実験成果報告 京都産業大学2

京都産業大学/Shibboleth2/Rails

京都産業大学に戻る

軽量アプリケーションでの Shibboleth 対応手順の調査（2008年度実証実験成果報告より）

- Wiki におけるアクセス制御
  - Hiki の紹介
  - Shibboleth (mod_shib) でのアクセス制御
  - Shibboleth (mod_shib) での Wiki アクセス制御における問題点
- Ruby on Rails でのアクセス制御
  - Ruby on Rails の紹介
  - redmine の紹介
  - Rails によるアプリケーションサーバの構築
  - Rails アプリケーションへの Shibboleth 認証の導入
    - Rails でのユーザ認証の実装
    - Rails での POST データ消失への対応
    - Rails でのユーザ認証への Shibboleth 認証の追加
    - Rails で Shibboleth 認証を利用する際の注意点
- プライバシを考慮したID受け渡し
  - transient-id による連携
  - persistent-id による連携
- まとめと今後の課題

Wiki におけるアクセス制御

ここでは軽量アプリケーションにおけるアクセス制御の例として，Ruby で実装された Wiki clone である Hiki を取り扱う．

Hiki の紹介

Hiki では，他の多くの Wiki 実装と同様に簡易なテキストにより，HTML ドキュメントを作成する機能を提供しており，また，plugin により機能拡張が可能になっている．デフォルトの動作では，管理者ユーザのみ存在しており，Wiki の動作を設定することができる．edit_user, auth_typekey プラグインを利用すると，ページの「新規作成」，「編集」の権限を登録したユーザあるいは TypeKey 認証されたユーザのみに限定することができる．

アプリケーション側では，「新規作成 (create)」，「編集 (edit)」の際に creatable?, editable? というメソッドを呼び出すことで，権限確認を行っている．edit_user, auth_typekey プラグインでは，これらのメソッドで呼び出されている auth? というメソッドをオーバーライドすることで，認証機能を拡張している．

Shibboleth (mod_shib) でのアクセス制御

Apache の mod_shib モジュールでは，Apache の <Location>, <LocationMatch> などのウェブ空間コンテナ を用いて Shibboleth 認証を実施する URL を指定する．ウェブ空間コンテナは，基本的に URL の protocol, host, port, path の部分を用いて対象を指定するため，クライアントが要求する URL に URL Query String が指定されていても，ウェブ空間コンテナの比較対象にはならない．

Shibboleth (mod_shib) での Wiki アクセス制御における問題点

Wiki の実装の詳細を考慮せずに，Wiki におけるコンテンツの更新に対して制限をかけるためには，直感的には以下のように Wiki URL に対して POST, PUT, DELETE メソッドを制限しておけば良いと考えられる．スクリプトで自動的に Web アクセスする (認証手順をあらかじめ把握している) 場合や，ユーザの利便性を考慮せずにコンテンツの保護のみ考える場合には，この方式でも要求を満すことができる．

```
% sudo vi /etc/httpd/extra/shibd.conf
<Location "/hiki">
 <Limit POST PUT DELETE>
   AuthType shibboleth
   ShibRequestSetting requireSession true
   Require valid-user
 </Limit>
</Location>
```

しかしこの方式では，認証されていないユーザがコンテンツの「編集」をクリックし，form に対して変更を入力した後，submit する時点で初めて認証がかかるため，編集したコンテンツ (POST したデータ) が Shibboleth のリダイレクトにより消失してしまう．また，アプリケーション (Wiki) 側で管理しているセッションと独立して Shibboleth 側でセッションを管理することになるため，アプリケーション側よりも Shibboleth 側のセッション有効期限が短かい場合には，同様な POST データの消失が発生する可能性がある．

このような意図しないリダイレクトによる POST データの消失を防ぐためには，POST データを編集する form 画面の取得時に認証を実行するのが望ましい．ここで，Hiki では例えば「編集」の form は

```
http://wiki.example.com/hiki/?c=edit;p=FrontPage
```

のような URL で取得できる．Hiki では「編集」をはじめとするコマンドを URL Query String に指定する形で実装されているため，ウェブ空間コンテナでは直接 URL 指定できない．URL Rewrite を行って Query String を path に変換するといった工夫が必要となる．

以上のようなことから，Wiki のような軽量アプリケーションにおいても，URL ベースで明確にリソースが指定できない場合は auth_typekey プラグインのように，アプリケーション側の認証機能として実装するのが望ましい．

Ruby on Rails でのアクセス制御

Ruby on Rails の紹介

Ruby on Rails とはプログラミング言語 Ruby を用いて MVC (Model View Controller) アーキテクチャに基づき Web アプリケーションを構築するためのフレームワークである．Ruby のスクリプト言語としての柔軟性を利用し，宣言的な機能追加などにより，アプリケーション開発者がアプリケーションロジックの記述に集中しやすい環境を提 供している．軽量アプリケーションの開発において採用されることが多い．

redmine の紹介

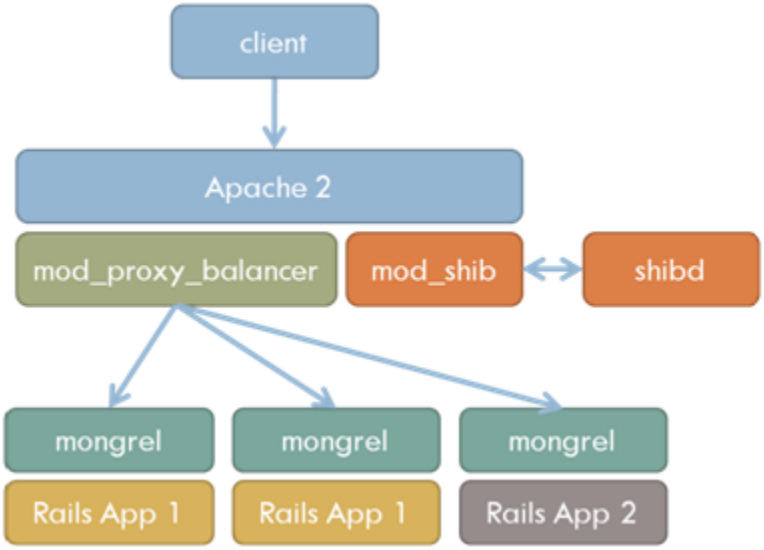
redmine とは Ruby on Rails で実装されたプロジェクト管理用の Web アプリケーションである．同様なアプリケーションとしては Python で実装された Trac が有名である．デモサイト (unofficial)で基本的な機能を体験できる．今回 Ruby on Rails での認証機能の実装を調査する上で，redmine の実装を参考にした．

Rails によるアプリケーションサーバの構築

Rails は，Ruby で実装されているため，fast_cgi, mod_ruby のようなモジュールを用いることで，Apache のような Web サーバ上で起動させることができる．しかし，これらのモジュールの安定性や性能が十分ではないため，Java における J2EE Container と同様に別途専用のアプリケーションサーバを起動して，その上で Rails アプリケーションを運用するケースが多い．

Rails 標準では，WEBrick という Ruby で実装された Web サーバ上でアプリケーションを動作させ，開発・デバッグを行うようになっているが，WEBrick は動作性能が高くないため，mongrel のようなより軽量なアプリケーションサーバが利用されることが多い．これらのアプリケーションサーバをやはり J2EE Container と同様に，Apache などのバックエンドとして動作させる形態が採用されている．

Rails は，2.1 以前はシングルスレッドで実装されており，1プロセスは同時に1リクエストしか処理できないため，複数プロセスを起動し，Apache2 の mod_proxy_balancer や Pound によりロードバランスさせる運用が良く用いられる．そのため，Shibboleth の認証を適用するには，Apache2 に mod_shib を導入して，mod_proxy_balancer で振り分けを行う前に認証をかけることになる (図1).



以下，Rails でのユーザ認証の実装方法，Rails で Shibboleth 認証を用いる場合の注意点について，mongrel をアプリケーションサーバとして利用した場合を例にあげて説明する．

Rails アプリケーションへの Shibboleth 認証の導入

Rails でのユーザ認証の実装

Rails アプリケーションでの認証の実装では restful-authentication plugin が比較的良く用いられている．restful-authentication では，認証処理に必要なモデル，ビュー，コントローラのテンプレートを生成する機能と，テンプレートから利用される認証機能のライブラリを提供している．

restful-authentication では，ユーザがアクセスしたコントローラで認証が必要な場合，Session コントローラにリクエストをリダイレクトし，ユーザが提供した「ユーザ名/パスワード」とデータベース上の認証情報 (User モデルに記録) を照合する．正しく照合できた場合はセッション情報として User モデルの id を記録し，元のページにリダイレクトする．セッション情報は暗号化された Cookie 内か，Cookie によって紐づけられたサーバ上のインスタンス (memory，DB 等) に保存される．コントローラ上からはセッション情報を参照できるため，一度認証されたユーザは同一アプリケーションの他の URL にアクセスしてもセッションを継続できる．

restful-authentication では，コントローラの action (クライアントからの URL リクエストに対して呼び出される Controller クラスのメソッド) に対して，before_filter (action の実行前に処理を挿入する機能) により認証処理を追加するので，基本的には action に対してアクセス制御することを想定している．redmine では restful-authentication の機能をモデルに対するアクセス制御も実施できるように拡張している．具体的にはコントローラにおいてセッションの情報をアクセスが発生したときに User モデルにアクセス中のユーザを記録している．モデルからコントローラで管理されているセッション情報を直接参照することはできないが，User モデルに保存することで他のモデルからも参照できるようにしている．

```
% vi app/controllers/application.rb
class ApplicationController < ActionController::Base
...
  before_filter :user_setup, ...
...
 def user_setup
   # Check the settings cache for each request
   Setting.check_cache
   # Find the current user
   User.current = find_current_user
 end

 # Returns the current user or nil if no user is logged in
 def find_current_user
   if session[:user_id]
     # existing session
     (User.active.find(session[:user_id]) rescue nil)
   elsif cookies[:autologin] && Setting.autologin?
     # auto-login feature
     User.find_by_autologin_key(cookies[:autologin])
   elsif params[:key] && accept_key_auth_actions.include?(params[:action])
     # RSS key authentication
     User.find_by_rss_key(params[:key])
   end
 end
```

なお，Rails ではすべての Controller クラスが ApplicationController を継承しているため，上記の実装はすべての Controller クラスで有効になる．以上のような実装により，例えばプロジェクトの情報のうち，特定のユーザに公開しても良い情報は visible_by メソッドで取得して表示するといった処理が記述可能になる．

```
% vi app/models/project.rb
class Project < ActiveRecord::Base
...
 def self.visible_by(user=nil)
   user ||= User.current
   if user && user.admin?
     return "#{Project.table_name}.status=#{Project::STATUS_ACTIVE}"
   elsif user && user.memberships.any?
     return "#{Project.table_name}.status=#{Project::STATUS_ACTIVE} AND (#{Project.table_name}.is_public = #{connection.quoted_true}
or #{Project.table_name}.id IN (#{user.memberships.collect{|m| m.project_id}.join(',')}))"
   else
     return "#{Project.table_name}.status=#{Project::STATUS_ACTIVE} AND #{Project.table_name}.is_public = #{connection.quoted_true}"
   end
 end
...
```

Rails においてアクセス制御を考える場合は，セッション情報を User モデルに一時保存して利用するのが望ましいと考えられる．これは他の MVC モデルのフレームワークでも同様であると考えられる．なお，Rails 2.2 以降ではスレッドに対応しているため，redmine では

```
% vi app/models/user.rb
class User < ActiveRecord::Base
...
 def self.current=(user)
   @current_user = user
 end

 def self.current
   @current_user ||= User.anonymous
 end
```

のようにクラスオブジェクトのインスタンス変数に保存しているが，複数のスレッドが同時にアクセスしてきた場 合，@current_user が上書きされてしまう可能性がある．そのため，以下のように Thread.current を用いてスレッドごとに変数を保存するのが望ましいと考えられる．

- 参考 URL: Thoughts on User.current and Thread Safety

```
% vi app/controllers/application.rb
class ApplicationController < ActionController::Base
 before_filter :current_user_id

 # Set the value
 def current_user_id
   Thread.current[:current_user_id] = session[:user_id]
 end
end
```

```
% vi app/models/user.rb
class User < ActiveRecord::Base
...
 def self.current
   Thread.current[:current_user] ||= self.fetch_by_id(Thread.current[:current_user_id])
 end
end
...
```

現在 redmine 0.9 で Rails 2.2 への対応が進められている．

Rails での POST データ消失への対応

Wiki におけるアクセス制御の 項目で述べた POST (PUT) リクエストの内容が認証のリダイレクトで消失してしまう問題について，Rails 自体の認証機能においてリダイレクトを利用しているので，Rails の認証において問題に対処できていれば，Shibboleth の認証は Rails 認証のサブモジュールとして実装できれば Shibboleth が直接問題の原因になることはない．

Rails の restful-authentication の認証では，before_filter を設定した特定の action に対してアクセスが発生した場合に認証処理が発生する (redmine の場合も同様)．Session コントローラにリダイレクトされる際には，元の URL の情報しかセッション情報として記録されないため，POST (PUT) でデータを送信していた場合には，データは失なわれる．しかし，before_filter を設定する対象として，実際に POST (PUT) を実行する URL 以外にユーザがデータを入力する form を返す URL に対しても before_filter を設定しておけば，認証はデータ入力前に実行されるため，POST (PUT) 時になって初めてリダイレクトが発生することはない．

以上のようなことから，Rails での Shibboleth 対応においては POST (PUT) によるデータ消失の問題は発生しないと考えられる．

- 注) 現状のブラウザ実装では PUT メソッドは用いられていないため，ユーザ操作により PUT リクエストが発生することはない．

Rails でのユーザ認証への Shibboleth 認証の追加

モデル，コントローラ双方でのアクセス制御が可能な redmine のユーザ認証に Shibboleth 認証の機能を追加した手順について説明する．

redmine ではユーザ登録機能があり，登録申請に対して管理者が承認したらユーザ登録が完了する．さらにシステムの管理者あるいはプロジェクトの管理者が必要に応じ て登録されたユーザをプロジェクトに追加する．また，redmine では外部の LDAP サーバを参照して認証する機能が提供されており，LDAP 上に存在するユーザについては，認証が成功すれば管理者の承認なしで登録できるようになっている．そこで，LDAP 認証機能を拡張する形で機能追加することにした．

LDAP 認証機能では AuthSource というモデルに参照する LDAP サーバの情報を登録している．この AuthSource に SSO サーバの情報も登録する形とし，また，他の SSO にも対応できるような実装とした．

Shibboleth 認証については，以下のような手順で導入できる．

1. ウェブ空間コンテナで指定できる URL を認証なしではアクセスできない「認証つき URL」とする

2. 「認証つき URL」において必要な属性情報をアプリケーション側に受け渡す
3. アプリケーション側では要求された属性情報を受け取った場合のみ認証成功とする

Shibboleth 用の認証 URL を用意し，属性が取得できたら既存のユーザ登録処理のコードを実行するように実装する (app/controllers/account_controller.rb および lib/sso_auth_methods.rb 参照)．Rails では URL と Controller の action のマッピングは Rails の ActionController の Routing 機能で実現されており，config/routes.rb にマッピングを追加する．

redmine 0.8-stable の r2252 に機能追加したものを以下のリポジトリに置いている．

```
git://gitosis.kyoto-su.ac.jp/redmine.git
```

以下のコマンドで r2252 からの変更点を確認できる．

```
% git clone git://gitosis.kyoto-su.ac.jp/redmine.git
% cd redmine
% git diff origin/base-svn master
```

また，以下の URL で Shibboleth 対応した redmine を仮運用している．

- Shibboleth 対応 redmine

現状の実装では単純に Rails で受け取った属性値 (氏名等) をアカウントの初期登録情報として利用しているだけだが，モデルへのアクセス制御に利用されている Role などへの対応付け機能を実装すれば，初期登録時から Shibboleth 属性に応じて適切なアクセス権限を付与することもできる．

Rails で Shibboleth 認証を利用する際の注意点

Apache + mod_proxy_balancer + mongrel の構成で Shibboleth を利用する場合，mod_shib のデフォルトの設定である「Apache の環境変数としてユーザ属性を受け渡す」では mongrel 側に属性値を受け渡すことができない．そのため，ユーザ属性を HTTP Header に設定するため，以下のように ShibUseHeaders On を追加する必要がある．

```
<Location /rails-application>
  AuthType shibboleth
  ShibRequestSetting requireSession true
  ShibUseHeaders On  # この記述が必要
  Require valid-user
</Location>
```

プライバシを考慮したID受け渡し

軽量アプリケーションのサーバは，研究目的で一時的に運用されるサーバなどが多く，また，継続運用される場合も学生による運用などがうまく引き継がれず，最悪の場合外部から侵入されてデータが漏洩する危険性がある．このような危険性を避けるために SAML SSO の連携範囲を限定するのは，教育研究活動支援の目的に逆行することになる．以下では SAML SSO 連携を実施する際に Shibboleth IdP 側の個人情報を保護しながら連携する方法について述べる．

transient-id による連携

Shibboleth IdP はデフォルトで SAML2.0 の transient-id による ID 受け渡しを利用する．transient-id の詳細については，SAML2.0 の仕様を参照のこと．

- 参考) Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0 の第8章の第3節

transient-id では，セッションごとに異なる ID を利用して IdP, SP 間で認証処理を実現するため，SP 側に記録された情報から IdP 側の個人を特定することはできない．そのため，必要最低限の属性情報のみ受け渡すことで，個人情報を保護することができる．

persistent-id による連携

Web アプリケーションによっては，同じユーザがアクセスしてきた場合，保存しておいた個人プロファイルを参照することで，前回の利用状況などを復元する処理が 必要になる．この場合，認証時に受け取る ID は同一人物に対しては毎回同じものでなければ個人特定に失敗してしまう．transient-id では，IdP 側の個人情報を保護することができるが，このような個人プロファイルを利用するアプリケーションに対応できない．

そこで，前述の SAML2.0 の仕様 で定義されている persistent-id を利用することになるが，Shibboleth では NameID の Format として persistent-id を実現するのではなく，Attribute Mapping の機能として実現している．具体的には eduPersonTargetedID を使うことで，persistent-id の機能を実現できる．以下では ComputedID を用いた簡易な persistent-id の利用方法および StoredID を用いた利用方法を紹介する．

なお，eduPersonTargetedID の詳細については FEIDE が解説を提供している．

- 参考) FEIDE による eduPersonTargetedID の解説

本家 Shibboleth のサイトでは，

- Computed ID Data Connector

の項目を参照のこと．

ComputedID では，SHA1 アルゴリズムを用いて，SP の entity ID，LDAP 上のユーザを一意に識別できる属性値 (デフォルトでは uid)，salt 値を用いて，SP ごとに暗号化された ID を提供する．毎回同じ ID を提供するため，persistent-id として利用できる．また，IdP の設定変更だけで利用できる．以下に利用手順を示す．

```
Shibboleth IdP 側での設定変更
### 以下の記述（コメントアウトされている）を有効にする.
% sudo vi /opt/shibboleth-idp/conf/attribute-resolver.xml
...
  <resolver:AttributeDefinition id="eduPersonTargetedID" xsi:type="SAML2NameID" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
nameIdFormat="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
       sourceAttributeID="computedID">
       <resolver:Dependency ref="computedID" />

       <resolver:AttributeEncoder xsi:type="SAML1XMLObject" xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
               name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" />
       <resolver:AttributeEncoder xsi:type="SAML2XMLObject" xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
               name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" friendlyName="eduPersonTargetedID" />
   </resolver:AttributeDefinition>
...
   <!-- Computed targeted ID connector -->
   <resolver:DataConnector xsi:type="ComputedId" xmlns="urn:mace:shibboleth:2.0:resolver:dc"
                           id="computedID"
                           generatedAttributeID="computedID"
                           sourceAttributeID="uid"
                           salt="your random string here">
       <resolver:Dependency ref="myLDAP" />
   </resolver:DataConnector>

### attribute-filter で eduPersonTargetedID を許可する.
% sudo vi /opt/shibboleth-idp/conf/attribute-filter.xml
   <AttributeFilterPolicy id="releaseTransientIdToAnyone">
...
       <AttributeRule attributeID="eduPersonTargetedID">
           <PermitValueRule xsi:type="basic:ANY" />
       </AttributeRule>

このとき releaseTransientIdToAnyone ではなく，以下のように限定した SP にのみ受け渡すようにしても良い.
   <AttributeFilterPolicy>
       <PolicyRequirementRule xsi:type="basic:AttributeRequesterString" value="https://sp01.auth.cmc.osaka-u.ac.jp/shibboleth"/>
      ...
       <AttributeRule attributeID="eduPersonTargetedID">
           <PermitValueRule xsi:type="basic:ANY" />
       </AttributeRule>
     ....
   </AttributeFilterPolicy>

### サーバ再起動
% sudo /etc/init.d/tomcat6 stop
% sudo /etc/init.d/tomcat6 start
```

```
SP 側の変更内容（特になし）
### デフォルトで以下の記述があるため
% sudo vi /etc/shibboleth/attribute-map.xml
   <!-- Third, the new version (note the OID-style name): -->
   <Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" id="persistent-id">
       <AttributeDecoder xsi:type="NameIDAttributeDecoder" formatter="$NameQualifier!$SPNameQualifier!$Name"/>
   </Attribute>
```

以上の設定により，環境変数 persistent-id で以下のような値が取得できる．

```
persistent-id=https://idp.example.org/idp/shibboleth!https://sp.sample.org/shibboleth!cEifP0U5XQ/H0ufyVAvyZ3x2P0g=
```

暗号化されているが，毎回同じIDが渡されるため個人特定に利用できる．

不正アクセス時などに ID を特定したい場合は，上記の ID がどのユーザのものであるか判定する必要がある．暗号化の手順は shibboleth-common-1.1.2.jar の

```
shibboleth-common-1.1.2/src/main/java/edu/internet2/middleware/shibboleth/common/attribute/resolver/provider/dataConnector
/ComputedIDDataConnector.java
```

に実装されている．

```
MessageDigest md = MessageDigest.getInstance("SHA");
md.update(inboundMessageIssuer.getBytes());
md.update((byte) '!');
md.update(sourceId.getBytes());
md.update((byte) '!');

computedIdAttrib.getValues().add(Base64.encodeBytes(md.digest(salt)));
```

inboundMessageIssuer には SP の entity ID，sourceId には attribute-resolver.xml で sourceAttributeID に定義した属性値 (uid)，salt には同じく attribute-resolver.xml で定義した salt の値が入るので，例えば Ruby で同じ ID を生成するには，

```
irb> require 'openssl'
irb> require 'base64'
irb> Base64.encode64(OpenSSL::Digest::SHA1.digest('https://sp.example.org/shibboleth!username!your random string here'))
=> "z214BxYlEWO/MO+AJ/nbQun6iSI=¥n"
```

つまり，

```
Base64.encode64(OpenSSL::Digest::SHA1.digest("#{sp_entity_id}!#{attribute}!#{salt}"))
```

で生成できる．確認するにはすべてのユーザについて一旦上記の変換を実施する必要がある．

本家 Shibboleth のサイトでは，

- Stored ID Data Connector

の項目を参照のこと．

Computed ID Data Connector のページに書かれているが，Computed ID Data Connector は非推奨の機能となっている．非推奨の理由は，

- IDの不正利用をトレースする際に一旦変換する手間がかかる．
- どのユーザがどのサイトと persistent-id が利用されているか確認できない．
- IDの生成がSHA1に依存しており一意性が保証されていない．

これに対し Stored ID Data Connector では，連携時には Database に persistent-id を保存できること，Type 4 UUID を persistent-id として利用できることなどから，今後は Stored ID の利用が推奨されている．以下では Stored ID の利用方法について述べる．なお，Database としては導入を容易にするため，大阪大学の成果「Shibboleth 2.0 の属性マッピング機能の検証」 で利用されている軽量 DB SQLite3 を導入例として扱う．

```
### SQLite3 の JDBC ドライバの取得
% wget http://files.zentus.com/sqlitejdbc/sqlitejdbc-v054-pure.jar
### SQLite3 の JDBC ドライバの IdP への導入
% cp sqlitejdbc-v054-pure.jar /opt/apache-tomcat-6.0.18/webapps/idp/WEB-INF/lib/
% cd /opt/shibboleth-idp/
% mkdir data
% cd data
### SQLite3 で Stored ID 用のデータベースの作成
% sqlite3 attributes.db
> CREATE TABLE shibpid (
   localEntity VARCHAR NOT NULL,
   peerEntity VARCHAR NOT NULL,
   principalName VARCHAR NOT NULL,
   localId VARCHAR NOT NULL,
   persistentId VARCHAR NOT NULL,
   peerProvidedId VARCHAR DEFAULT NULL,
   creationDate TIMESTAMP NOT NULL,
   deactivationDate TIMESTAMP NULL
);
> CREATE INDEX shibpid_persistentId_index ON shibpid(persistentId);
> CREATE INDEX shibpid_persistentId_deactivationDate_index ON shibpid(persistentId, deactivationDate);
> CREATE INDEX shibpid_localEntity_peerEntity_localId_index ON shibpid(localEntity, peerEntity, localId);
> CREATE INDEX shibpid_localEntity_peerEntity_localId_deactivationDate ON shibpid(localEntity, peerEntity, localId, deactivationDate);
> .quit

### attribute-resolver の設定変更
### computeID を storedID に変更
% emacs /opt/shibboleth-idp/conf/attribute-resolver.xml
   <!-- eduPersonTargetedID の参照先を storedID に変更 -->
   <resolver:AttributeDefinition id="eduPersonTargetedID" xsi:type="SAML2NameID" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
       nameIdFormat="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
       sourceAttributeID="storedID">
       <resolver:Dependency ref="storedID" />

       <resolver:AttributeEncoder xsi:type="SAML1XMLObject" xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
               name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" />

       <resolver:AttributeEncoder xsi:type="SAML2XMLObject" xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
               name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" friendlyName="eduPersonTargetedID" />
   </resolver:AttributeDefinition>

   <!-- Stored ID Data Connector の設定 -->
   <resolver:DataConnector xsi:type="StoredId" xmlns="urn:mace:shibboleth:2.0:resolver:dc"
                           id="storedID"
                           generatedAttributeID="storedID"
                           sourceAttributeID="uid"
                           salt="your random string here">
       <resolver:Dependency ref="myLDAP" />

       <ApplicationManagedConnection jdbcDriver="org.sqlite.JDBC"
               jdbcURL="jdbc:sqlite:/opt/shibboleth-idp/data/attributes.db" />

   </resolver:DataConnector>

### サーバ再起動
% sudo /etc/init.d/tomcat6 stop
% sudo /etc/init.d/tomcat6 start

### ブラウザで初回ログイン時に DB 上に persistent-id が記録される.

### persistent-id および連携しているサイトの確認方法
% cd /opt/shibboleth-idp/data
% sqlite3 attributes.db
> SELECT * FROM shibpid;
...
```

Stored ID Data Connector では，データベース上の ID をそのまま利用するので，すでに Computed ID で連携構築している場合は，その ID を DB 上に格納しておくことで，ユーザおよび連携先にはシームレスに移行することも可能である.

まとめと今後の課題

Shibboleth SP の実装を用いて軽量アプリケーションに対するアクセス制御方法を調査した結果，ユーザの利便性を保つためには簡単な Wiki アプリケーションにおいても注意が必要であることが確認できた．また，Ruby on Rails のような Web アプリケーションフレームワークにおけるユーザ認証の実装について調査することで，MVC モデルにおけるユーザ認証，アクセス制御の実現方法の基礎について理解することができた．今後の課題として，より簡易に認証，アクセス制御機能が導入でき るように，restful-authentication のような plugin として SSO 対応の認証機能を提供していくことなどが考えられる．

京都産業大学に戻る