La démarche d'intégration de Shibboleth

dans un service en ligne ou une application *web*

Florent Guilleux
Comité Réseau des Universités
CRU – CRI, 263 avenue du Général Leclerc, CS 74205, campus de Beaulieu 35042 Rennes CEDEX
florent.guilleux@cru.fr

Olivier Salaün
Comité Réseau des Universités
CRU – CRI, 263 Avenue du Général Leclerc, CS 74205 campus de Beaulieu 35042 Rennes CEDEX
olivier.salaun@cru.fr

Résumé

Cet article propose une démarche structurée pour rendre compatible un service en ligne ou une application web avec Shibboleth. Une grille d'analyse du contexte d'utilisation de l'application permet d'abord de définir les modalités de Shibboleth à mettre en œuvre en termes d'identifiants, d'attributs et de contrôle d'accès notamment. La seconde partie présente le travail de « shibbolisation » proprement dit du code d'une application. Outre une présentation du fonctionnement technique d'un fournisseur de services, sont détaillées différentes approches de gestion de l'authentification, de l'autorisation, des comptes utilisateurs, des sessions et du logout.

Mots clefs

Shibboleth, fédération d'identités, authentification, autorisation, SSO, contrôle d'accès, développement

1 Introduction

L'usage de Shibboleth pour simplifier et sécuriser l'accès à des ressources *web* partagées entre établissements se développe dans les communautés enseignement supérieur et recherche, en France et à l'étranger. Les fédérations d'identités et le fonctionnement de Shibboleth ont déjà été présentés aux JRES [1]. Pour qu'une application *web* ou un service en ligne soit accessible via Shibboleth, il faut les rendre compatible avec Shibboleth (cette opération est désignée dans cet article par « shibboliser une application »). Ce travail est similaire dans ses objectifs à celui de « CASsification » d'une application : on souhaite s'affranchir du mode natif d'authentification de l'application pour une authentification extérieure. Cependant l'architecture de Shibboleth est plus complexe que celle de CAS ; ses modalités et ses contextes d'utilisation sont plus variés. L'objectif de cet article est de présenter aux développeurs devant « shibboliser » une application les éléments leur permettant de choisir quelles fonctionnalités de Shibboleth ils utiliseront, et comment les implémenter.

Cet article ne traite pas de l'installation et de la configuration de la brique fournisseur de services Shibboleth. Une connaissance des principes de base des fédérations d'identités et de Shibboleth est souhaitable pour la lecture de cet article. Cet article a été rédigé sur la base de l'expérience du CRU dans la gestion de son service de fédération[1] et la « shibbolisation » des applications Sympa, DokuWiki, GForge et de modules d'authentification pour Apache.

On utilisera le terme « application » pour désigner indifféremment une application ou un service en ligne. Pour rappel, Shibboleth ne fonctionne qu'avec des applications accessibles via un navigateur *web*.

Une version plus détaillée de cet article et qui sera régulièrement mise à jour est disponible sur le site web du CRU[2].

2 Cerner le contexte d'utilisation de l'application

L'adaptation technique d'une application pour la rendre compatible avec Shibboleth est décrite dans le troisième chapitre. Ce travail doit être précédé d'une analyse du contexte de l'utilisation de l'application dans un cadre de fédération d'identités. Cela va permettre d'identifier notamment quelles fonctionnalités de Shibboleth seront utilisées, les pré-requis que devront satisfaire les fournisseurs d'identités, le modèle de contrôle d'accès retenu, etc.

2.1 Couvrir toute la population utilisatrice

Tous les utilisateurs de l'application sont-ils rattachés à un organisme qui est fournisseur d'identités ? c.-à-d. tous les utilisateurs pourront-ils utiliser Shibboleth pour accéder à l'application ? Si ce n'est pas le cas, il y a deux solutions pour gérer les utilisateurs qui ne sont pas rattachés à un établissement fournisseur d'identités.

[1] http://federation.cru.fr

[2] http://www.cru.fr/faq/federation/shibbolisation

La première solution est de maintenir un système d'authentification interne à l'application qui fournira un moyen de s'authentifier à ces utilisateurs. L'application proposera aux utilisateurs de s'authentifier soit via ce compte local, soit via Shibboleth pour ceux qui sont rattachés à un organisme qui est fournisseur d'identités.

La seconde solution, permettant de s'affranchir d'un mode d'authentification mixte, est d'ajouter à la liste des fournisseurs d'identités un fournisseur d'identités « virtuel ». C'est un fournisseur d'identités qui n'est pas rattaché à un organisme réel, mais qui peut offrir des comptes utilisateurs aux personnes qui ne sont pas rattachées à un organisme disposant d'un fournisseur d'identités Shibboleth. Cela permet d'utiliser Shibboleth comme seul mode d'authentification ; ainsi on simplifie l'ergonomie pour les utilisateurs et l'architecture de l'application. On peut soit utiliser un fournisseur d'identités virtuel déjà opérationnel, par exemple le Service d'Authentification du CRU[3], soit installer un service similaire dédié à son application ou à un ensemble d'applications (par exemple dans le cadre d'une UNR).

2.2 Identifier les utilisateurs

Quand une application authentifie elle-même ses utilisateurs, elle peut facilement utiliser un identifiant local et spécifique à l'application.

Quand une application « shibbolisée » doit identifier ses utilisateurs, elle doit utiliser un identifiant fourni par le fournisseur d'identités quand l'utilisateur se connecte. Elle n'utilise pas l'identifiant saisi par l'utilisateur sur son SSO (uid ou supannAliasLogin) car comme les utilisateurs sont issus de différents organismes, il n'y a pas garantie de l'unicité de ces identifiants.

Un identifiant nominatif doit être pérenne et unique dans la population des utilisateurs de l'application. L'identifiant nominatif qui respecte le mieux ces contraintes est l'identifiant institutionnel (*eduPersonPrincipalName)* défini dans la recommandation SUPANN [2]. Une alternative est d'utiliser l'adresse email de l'utilisateur qui a l'avantage d'être déjà très utilisée. Cependant il faut noter qu'une adresse email n'est a priori pas pérenne (changement de nom ou de domaine d'établissement).

Dans certaines situations, on souhaite disposer d'un identifiant, sans pour autant qu'il soit nominatif. Par exemple quand l'application n'a pas besoin de connaître l'identité précise d'une personne mais uniquement certains attributs non nominatifs pour le contrôle d'accès.

Si l'application souhaite pouvoir reconnaître l'utilisateur d'une session à une autre, elle peut utiliser l'attribut *eduPersonTargetedId* qui est un identifiant utilisateur opaque, persistant et spécifique à un couple fournisseur d'identités / fournisseur de services. Cependant elle nécessite l'installation d'un *plugin* au niveau des fournisseurs d'identités dans Shibboleth 1.3. L'utilisation simple de cet attribut ne sera possible que dans Shibboleth 2.0.

Shibboleth fournit toujours à une application le *NameIdentifierFormat* ShibHandle qui est un identifiant utilisateur de session opaque, temporaire et généré automatiquement par un fournisseur d'identités Shibboleth[4]. Il change à chaque session et ne contient aucune information nominative sur l'utilisateur.

Tous ces identifiants utilisateur sont mis à disposition de l'application par Shibboleth sous forme d'en-têtes HTTP, comme les autres attributs. Dans la fédération du CRU le nom des attributs disponibles est normalisé[5]. Un fournisseur de services devra s'assurer auprès des fournisseurs d'identités qu'ils sont prêts à diffuser l'identifiant retenu.

2.3 Contrôler l'accès

Pour certains services en ligne, le simple fait qu'un utilisateur soit authentifié auprès d'un fournisseur d'identités reconnu suffit à donner l'accès aux contenus du service. Par exemple dans la fédération du CRU toute personne s'authentifiant dans un établissement reconnu par l'éditeur *Elsevier* a accès au portail *ScienceDirect*, car elle est considérée comme membre d'un établissement d'enseignement supérieur.

Cependant la plupart des applications contrôlent l'accès des utilisateurs après leur authentification. En général cette opération de contrôle d'accès consiste à appliquer des règles d'autorisation basées sur des attributs décrivant l'utilisateur pour déterminer ses privilèges applicatifs. Le plus souvent ces règles de contrôle d'accès sont définies et exécutées localement au niveau de l'application[6], soit directement dans le code applicatif soit au niveau de la brique Shibboleth fournisseur de services.

Voici quelques exemples :

1. Accès à une plate-forme d'enseignement à distance
   - si mail = 'dupont@univ-xx.fr' alors accès avec droit d'administration ;
   - sinon
     - si eduPersonAffilation = 'faculty' alors accès à l'application avec droit d'écriture,
     - si eduPersonAffiliation = 'student' alors accès à l'application avec droit de lecture,
     - sinon accès refusé,
2. Accès au wiki d'un groupe de travail
   - si email appartient à {liste d'emails définie dans l'application et définissant le groupe} alors accès à l'application,

[3] http://www.cru.fr/faq/federation/service-authentification-cru

[4] SAML 2.0 a standardisé cette notion d'identifiant de session opaque avec le format *transient*. Dans Shibboleth 2.0 ce nouveau format remplace *ShibHandle*.

[5] http://federation.cru.fr/cru/attributs.html

[6] Le standard XACML définit une architecture d'autorisation centralisée, hors des applications, mais il est encore peu répandu (http://www.oasis-open.org/committees/xacml/).

- sinon accès refusé,

3. Accès à un portail documentaire
   - si eduPersonEntitlement = 'common-libs-terms' alors accès à l'application,
   - sinon accès refusé,

4. Accès à une plate-forme de dépôt de documents pour un groupe de travail :
   - si email appartient à {liste d'emails définie dans une base extérieure et définissant le groupe} alors accès à l'application ,
   - sinon accès refusé.

Dans l'exemple (1) l'application est entièrement maître du contrôle d'accès car elle détermine selon quels profils génériques l'accès est réalisé.

Idem avec l'exemple (2) où elle s'appuie sur une liste énumérée d'utilisateurs qu'elle stocke en interne.

Dans l'exemple (3) elle s'appuie sur la valeur *common-lib-terms* de l'attribut *eduPersonEntitlement*. Cette valeur est positionnée par les fournisseurs d'identités pour chacun de leurs utilisateurs qui satisfont à la sémantique de cette valeur. L'application fait confiance au positionnement correct de cette valeur par les fournisseur d'identités. Dans un sens, l'application délègue donc la maîtrise du contrôle d'accès aux fournisseurs d'identités.

Dans l'exemple (4), pour son contrôle d'accès, l'application s'appuie sur une liste énumérée d'utilisateurs définie dans une base extérieure. Par exemple cela peut être le cas pour un groupe d'utilisateurs répartis entre différents organismes qui constituent ce que l'on appelle couramment une « organisation virtuelle » (*Virtual Organization*). La liste des membres de ce groupe est gérée dans un référentiel extérieur (annuaire LDAP, *Virtual Organization manager*, serveur de listes de diffusion, etc.) par une personne qui n'est pas nécessairement le gestionnaire de l'application. D'autre part d'autres applications d'intérêt pour le groupe s'appuient peut-être sur cette base pour leur propre contrôle d'accès. Il n'est donc pas souhaitable de gérer cette liste dans chacune de ces applications ; mais plutôt que chaque application concernée interroge le référentiel pour ses propres besoins (dynamiquement ou en maintenant une copie locale de ce référentiel).

Le choix de tel ou tel modèle pour les règles de contrôle d'accès d'une application dépend à la fois des contraintes de disponibilité et localisation d'attributs et des accords sur la diffusion et la définition d'attributs entre l'application fournisseur de services et les fournisseurs d'identités.

2.4 Déterminer d'autres besoins en attributs

Les attributs utilisateurs peuvent être utilisés à d'autres fins que le contrôle d'accès. Par exemple on peut utiliser l'attribut *preferredLanguage* pour adapter la langue d'affichage de l'application, le *displayName* pour avoir un message d'accueil personnalisé, le profil (*eduPersonAffiliation*, *supannAffectation*, etc.) pour mettre en valeur sur la page d'accueil certains contenus d'intérêt pour l'utilisateur, des informations nominatives (*givenName*, *sn*, *telephoneNumber*) pour renseigner automatiquement le compte local de l'utilisateur.

Il faut déterminer avec les fournisseurs d'identités quels sont les attributs qu'ils sont prêts à fournir, et quels attributs sont obligatoires et optionnels.

2.5 Éventuellement pré-alimenter des comptes utilisateurs locaux

Shibboleth ne permet de récupérer un identifiant et des attributs sur un utilisateur que lorsque que ce dernier accède effectivement à l'application.

Parfois il est nécessaire que des informations sur des utilisateurs soient renseignées dans certaines applications avant que ces utilisateurs n'accèdent à l'application. Par exemple sur une plate-forme d'enseignement à distance, un enseignant se connectant avant la rentrée peut souhaiter consulter la liste des étudiants de sa filière. Ces derniers ne s'étant pas déjà connectés, il faut prévoir au préalable l'approvisionnement de leurs identités dans l'application.

Shibboleth ne gérant pas ce mécanisme, il faut mettre en œuvre un mécanisme *ad hoc*.

3 « Shibboliser » une application

Ce chapitre décrit la démarche technique permettant de rendre une application compatible avec Shibboleth. Le résultat d'un tel travail peut être que :

(a) l'accès à l'application n'est désormais possible que via Shibboleth ;

(b) l'application peut être configurée avec un seul type d'accès simultanément : soit via Shibboleth, soit via la méthode d'authentification pré-existante ;

(c) l'accès à l'application est possible à la fois via Shibboleth et la méthode d'authentification pré-existante. Cela permet d'avoir une application dont une partie des utilisateurs s'authentifie via Shibboleth et une autre partie via un autre mode d'authentification.

3.1 Démarche en fonction du type d'application

Le scénario le plus simple est celui d'une nouvelle application, l'intégration avec Shibboleth pouvant être prévue dans les spécifications. Dans les autres cas, il sera nécessaire de modifier l'application. Une difficulté supplémentaire existe pour la « shibbolisation » d'un service déjà en production, où il faut prévoir un mécanisme de transition entre l'existant et l'utilisation de Shibboleth.

Si vous avez accès au code et le droit de le modifier, vous pouvez modifier l'application pour la rendre compatible avec Shibboleth. Voici la démarche à suivre :

1. vérifier si l'application n'a pas déjà été « shibbolisée »[7] ;

[7] https://wiki.internet2.edu/confluence/display/seas/Home

2. étudier le code, les API disponibles, les paramètres de configuration de l'application. Il est essentiel de bien comprendre sa gestion :
   - de l'authentification,
   - de l'autorisation,
   - des comptes utilisateurs,
   - du *logout* et des sessions de connexion ;
2. choisir un mode d'utilisation du WAYF (voir « 3.7 Articulation avec un WAYF ») ;
2. si l'application est *open source*, exposer le projet d'extension aux auteurs de l'application ;
3. l'implémenter ;
4. le tester ;
5. si l'application est *open source*, soumettre un *patch* (ou mieux un *plugin* si le produit le permet) et sa documentation aux auteurs.

Dans le cas d'une application pour laquelle vous ne disposez pas du code source, le travail d'intégration pourra être plus complexe ; il nécessitera le cas échéant des développements à façon de la part de l'éditeur.

3.2 Principe de fonctionnement du *Service Provider* Shibboleth

La brique *fournisseur de services* de Shibboleth est un module Apache ou un filtre ISAPI pour IIS. Une autre implémentation du fournisseur de services en Java est en cours de mise au point. Cette brique gère en frontal de l'application toute la séquence Shibboleth : redirection vers le WAYF (qui redigire l'utilisateur vers son établissement, voir « 3.7 Articulation avec un WAYF »), récupération du jeton d'authentification une fois l'authentification effectuée au niveau du fournisseur d'identités, récupération des attributs utilisateurs. L'application est donc déchargée de tout ce traitement.

L'API qu'utilise un fournisseur de services Shibboleth pour transmettre les identités et les attributs à une application est relativement simple et générique. Le module Shibboleth met les attributs utilisateur à disposition dans des variables d'environnement CGI (contexte Apache) ou dans les en-têtes HTTP (contexte IIS ou J2EE), qui sont récupérables par l'application. Ce fonctionnement est similaire à celui d'autres modules d'authentification pour Apache, comme mod_ssl qui met à disposition tous les attributs d'un certificat X.509.

Les attributs envoyés à un fournisseur de services sont d'abord filtrés par l'ARP (*Attribute Release Policy*) au niveau des fournisseurs d'identités, puis au niveau du fournisseur de services ils sont filtrés via l'AAP (*Attribute Acceptance Policy*) avant d'être mis à disposition de l'application. Il est donc possible que certains attributs attendus par l'application ne soient pas transmis, du fait d'erreurs de configuration ou par choix d'un fournisseur d'identités. L'application doit donc traiter les cas de figure où les attributs escomptés ne sont pas transmis.

3.3 Stratégies d'implémentation de l'authentification

Nous allons étudier les différentes approches permettant à une application d'exploiter une authentification Shibboleth. Elles impliquent une configuration du fournisseur de services Shibboleth et/ou des modifications dans l'application. Le choix de telle ou telle approche dépend à la fois de l'architecture de l'application concernée et du niveau d'intégration souhaité.

Détournement d'un autre mode d'authentification

L'application est peut-être déjà capable d'exploiter un mécanisme d'authentification externe (authentification Apache, authentification LDAP, authentification par certificats X.509). Dans certains cas il sera possible de configurer l'application dans ce mode de fonctionnement pour faire fonctionner Shibboleth et éviter ainsi de modifier l'application. Par exemple, une application exploitant une adresse email issue d'un certificat personnel et transmise par mod_ssl pourra facilement traiter la même information si elle est transmise via Shibboleth.

Cette approche, comme les suivantes, nécessite malgré tout la configuration d'un fournisseur de services Shibboleth.

Toute l'application est protégée par Shibboleth

Il s'agit de la solution la plus simple pour protéger une application avec Shibboleth. On définit dans la configuration du serveur *web*, en amont de l'application, que tout accès requiert une authentification via Shibboleth. Ci-dessous, un exemple d'extrait de configuration Apache activant ce mode de fonctionnement :

```
<Location /mon-appli>
  AuthType shibboleth
  ShibRequire Session On
  require valid-user
</Location>
```

L'inconvénient de cette solution réside dans son manque de souplesse. En effet, elle ne permet pas de maintenir une partie de l'application accessible à des personnes non authentifiées, par exemple pour un wiki dont une partie des pages est accessible publiquement. Par ailleurs cette solution n'est pas utilisable si l'on souhaite que l'application propose plusieurs méthodes d'authentification aux utilisateurs.

Cette solution est adaptée pour contrôler l'accès à des contenus statiques ou pour des applications nécessitant une authentification en préalable à toute opération.

Une partie de l'application est protégée par Shibboleth, configuration d'Apache

Cette approche n'est utilisable que pour une application dont la phase d'authentification transite par un script dédié (par exemple login.php dans la configuration ci-dessous). Cette méthode permet donc de ne déclencher l'authentification Shibboleth qu'au moment voulu (*login*), tout en laissant le reste de l'application accessible anonymement.

Voici un exemple de configuration Apache pour le faire :

```
<Location /mon-appli/login.php>
  AuthType shibboleth
  ShibRequire Session On
  require valid-user
</Location>
```

Une partie de l'application est protégée par Shibboleth, utilisation des lazy sessions

Dans toutes les situations où on veut ne protéger qu'une partie de l'application mais où on ne peut pas appliquer la méthode précédente, on peut utiliser le mécanisme des *lazy sessions* proposé par Shibboleth : il permet à une application de déclencher la phase d'authentification Shibboleth à l'initiative de l'application. L'activation du module d'authentification Shibboleth (notion de *session initiator* dans Shibboleth) sera déclenchée par l'accès à une URL locale ayant un format spécifique.

Le renvoi de l'utilisateur vers l'URL d'initiation de session peut être pris en charge par l'application (via une directive HTTP de redirection) ou bien au niveau du serveur *web*, en utilisant les fonctions de réécriture d'adresses. Cette dernière solution permet de limiter les modifications à apporter à l'application, mais disperse les modifications apportées entre l'application et la configuration du serveur *web*.

Voici l'exemple de ce qui a été réalisé pour la « shibbolisation » de DokuWiki. Nativement, dans cette application, quand un utilisateur clique sur le bouton '*login*', la page courante est rechargée avec le paramètre CGI 'do=login' et une bannière de *login* est affichée dans la page. La directive Apache ci-dessous permet d'intercepter toute requête contenant 'do=login' pour redigirer l'utilisateur vers une URL activant le mode *lazy session.* L'URL initiale est passée en paramètre pour permettre le retour ultérieur vers l'application.

```
RewriteCond %{QUERY_STRING} "do=login"
RewriteRule ^/(.*)
https://sp.cru.fr/shibboleth/Shibboleth.sso/
WAYF/?target=%{HTTP_REFERER} [R]
```

L'utilisation du mécanisme des *lazy sessions* semble donc offrir les meilleures perspectives d'intégration pour une application utilisant Shibboleth.

Utilisation d'un reverse proxy

Cette approche peut être la seule envisageable dans le cas d'une application propriétaire où aucune méthode d'authentification existante ne peut s'adapter à Shibboleth et si l'option d'un développement à façon n'est pas possible.

Dans cette approche l'accès au *reverse proxy* est contrôlé par un fournisseur de service Shibboleth, et le *reverse proxy* mime une authentification attendue par l'application. Du point de vue de l'utilisateur le *reverse proxy* est une ressource « shibbolisée » nécessitant une phase d'authentification ; du point de vue de l'application la requête de *login* semble venir d'un utilisateur, alors qu'elle transite par le *reverse proxy*. Ce dernier aura peut-être des traitements à effectuer, par exemple pour faire correspondre le type d'identifiant remonté par Shibboleth à un identifiant attendu par l'application.

Pour des raisons de sécurité, il est essentiel de s'assurer que le *reverse proxy* est le seul point d'accès à l'application.

Utilisation de bibliothèques clientes SAML

Par analogie avec la « CASification » d'une application, il serait envisageable de s'appuyer sur des bibliothèques clientes SAML [3] (protocole utilisé entre le fournisseur d'identités et le fournisseur de services Shibboleth) pour rendre une application compatible avec Shibboleth. Cependant cette approche n'est pas réalisable avec Shibboleth 1.x, parce qu'il n'existe pas (encore ?) de bibliothèques clientes.

Shibboleth 2.0 est compatible avec SAML 2.0, il est donc possible d'utiliser des bibliothèques SAML 2.0 pour « shibboliser » des applications pour Shibboleth 2.0.

3.4 Stratégies d'implémentation de l'autorisation

Shibboleth est en premier lieu un système d'authentification, même s'il propose des fonctionnalités basiques de contrôle d'accès. En fonction des besoins de chaque application, on pourra utiliser les fonctionnalités de Shibboleth ou prendre en charge tout le processus de contrôle d'accès dans l'application.

Gestion du contrôle d'accès par Shibboleth

Le fournisseur de services Shibboleth propose des directives de contrôle d'accès dont la syntaxe est proche de celle du mode d'authentification natif du serveur Apache. La syntaxe proposée permet de tester la valeur d'un ou de plusieurs attributs utilisateur transmis par le fournisseur d'identités. Voir l'exemple ci dessous :

```
AuthType shibboleth
ShibRequireSession On
ShibRequireAll On
require affiliation student
require homeOrganization ~ ^univ-test.fr$
```

Alternativement, Shibboleth offre la possibilité d'exprimer les règles de contrôle d'accès au format XML.

Gestion du contrôle d'accès par l'application

Lorsque les règles de contrôle d'accès ne sont pas exprimables par quelques règles en dehors de l'application, on préférera gérer ce processus dans le code de l'application. Cette solution offre plus de souplesse quant à la gestion des erreurs ; elle permet également d'exploiter pleinement d'éventuels mécanismes d'ACL utilisés par l'application.

```
if ($ENV{‘HTTP_SHIB_EP_AFFILIATION’} == ‘student’
&&  $ENV{'HTTP_SHIB_SUPANN_SUPANNORGANISME'}  ==
'{ILN}1-341725201')

       { &authorizeUser() }
```

Dans le cadre de la gestion du contrôle d'accès, les attributs utilisateur transmis par le fournisseur d'identités peuvent se révéler insuffisants pour prendre une décision d'autorisation. Dans ce cas, il est tout à fait envisageable que l'application interroge une source de données tierce (annuaire LDAP, base SQL) afin de compléter le profil de l'utilisateur. À l'avenir, le fournisseur de services

Shibboleth inclura une fonctionnalité *attribute resolver* permettant d'interroger des sources de données externes (distinctes du fournisseurs d'identités) et donc d'enrichir les attributs utilisateur en amont d'une application.

3.5 Gestion de la base interne des utilisateurs

La quasi totalité des applications ont une base interne où sont recensés les utilisateurs, avec leur identifiant et des attributs les décrivant (nom, prénom, etc.) ou leur conférant des privilèges spécifiques à l'application. En général il n'est pas possible, lorsqu'on « shibbolise » une application, de supprimer cette base. En effet elle reste indispensable s'il faut gérer les privilèges applicatifs spécifiques à l'application. Également sa suppression nécessiterait en général trop de modifications à apporter au code et à son architecture.

Aussi il faut prévoir comment gérer cette base quand les identités et éventuellement des attributs sont propagés depuis l'extérieur, via Shibboleth, vers l'application. Par exemple :

- si nécessaire, remplacer le format de l'identifiant utilisateur local dans la base par celui remonté par Shibboleth, ou mettre en place une table de correspondance entre les identifiants locaux de l'application et les identifiants remontés par Shibboleth.
- supprimer le champ 'mot de passe' ou à défaut le rendre inutilisable en le positionnant à une valeur par défaut. Supprimer toute l'interface de gestion du mot de passe (rappel, modification).
- à la connexion d'un utilisateur via Shibboleth, vérifier si celui-ci est déjà dans la base interne de l'application :
  - si oui, vérifier que les attributs de l'utilisateur remontés par Shibboleth sont identiques à ceux de la base interne (par exemple le nom ou l'email) et si nécessaire mettre la base à jour ;
  - sinon créer le compte dans la base, si possible automatiquement, sinon en proposant un formulaire à l'utilisateur dans lequel il renseigne des informations nécessaires à la base et qui ne sont pas remontées par Shibboleth
- empêcher dans l'interface de gestion de compte de l'application la modification des attributs remontés via Shibboleth qui sont enregistrés dans la base.

Pour les services déjà en production que l'on souhaite « shibboliser », il faut prévoir un mécanisme de transition entre la gestion purement locale des comptes et le fonctionnement avec Shibboleth.

3.6 Sessions et gestion du *logout*

L'architecture distribuée de Shibboleth implique la mise en place de plusieurs sessions avec l'utilisateur, à différents niveaux. Ces sessions sont généralement maintenues au moyen de *cookies* HTTP :

1. session avec le SSO : Shibboleth n'implémentant pas directement de mécanismes d'authentification (cette situation change dans Shibboleth 2.0), il repose sur une application tierce, un serveur CAS par exemple, qui gère une session d'authentification avec l'utilisateur.
2. session avec le fournisseur d'identités Shibboleth : le fournisseur d'identités maintient une session pour éviter de solliciter le serveur CAS, à chaque connexion de l'utilisateur.
3. session avec le fournisseur de services : une fois l'utilisateur authentifié, le fournisseur de services maintient une session, pour ne pas recontacter le fournisseur d'identités à chaque requête de l'utilisateur.
4. session applicative : l'application maintient généralement une session d'authentification.

Le fournisseur de services Shibboleth fournit une URL de *logout*, charge à l'application d'orienter l'utilisateur vers celle-ci après la déconnexion de l'application et la suppression de la session applicative. Dans l'idéal, l'opération de *logout* de Shibboleth devrait se propager jusqu'au serveur CAS et même entraîner une déconnexion de toutes les autres applications ayant utilisé le mécanisme Shibboleth. Dans les versions 1.x, le fournisseur de services Shibboleth ne propage pas le *logout*. La version 2.0 de Shibboleth implémente cette propagation du *logout* (« *single logout* »).

3.7 Articulation avec un WAYF

Lorsqu'un utilisateur accède à un fournisseur de services Shibboleth, ce dernier ne sait pas a priori vers quel fournisseur d'identités le redigirer. Aussi l'utilisateur doit fournir cette information à la brique WAYF (« Where Are You From? »). Le WAYF consiste généralement en un menu déroulant où l'utilisateur peut sélectionner son établissement de rattachement pour y être ensuite redirigé.

Un WAYF doit être mis à disposition d'une application « shibbolisée ». Plusieurs possibilités existent :

- se reposer sur un WAYF déjà en exploitation, par exemple celui utilisé dans le cadre d'une UNR ;
- installer un WAYF dédié à l'application. Il existe plusieurs implémentations[8] ;
- implémenter le code d'un WAYF directement dans l'application. On peut ainsi intégrer le menu

[8] http://www.cru.fr/faq/federation/questions_reponses_pour_les_wayf#quelle_implementation_du_wayf_dois-je_utiliser

déroulant des fournisseurs de services directement dans la bannière de *login* de l'application.

Shibboleth offre par ailleurs la possibilité à l'utilisateur de contacter directement un fournisseur d'identités, en spécifiant le fournisseur de services à contacter. Cette méthode permet à un établissement de pointer vers des ressources, depuis son portail par exemple, en court-circuitant l'étape WAYF[9].

4 Conclusion

L'analyse du code pré-existant d'une application est le travail le plus significatif dans la démarche de « shibbolisation » d'une application. L'autre pré-requis important est la compréhension du fonctionnement de Shibboleth et la connaissance des possibilités de paramétrage de son fournisseur de services. Finalement la modification proprement dite du code est généralement peu importante.

Shibboleth bénéficie d'une communauté active et dynamique : de plus en plus d'applications compatibles avec Shibboleth sont disponibles [4]. Il ne faut pas hésiter à faire connaître son travail de « shibbolisation » d'une application pour contribuer à la mutualisation des efforts.

Les services *middleware* de type LDAP, CAS et maintenant Shibboleth se généralisent. La capacité d'intégration d'une application dans de telles architectures prend donc de l'importance. On peut espérer que les développeurs prennent de plus en plus en compte cette problématique dès la conception de leurs applications.

Bibliographie

[1] Pascal Aubry, Olivier Salaün, Florent Guilleux, Fédération d'identités et propagation d'attributs avec Shibboleth. *Tutoriel JRES2005*, Marseille, décembre 2005. *http://2005.jres.org/tutoriel/shibboleth-jres2005-article.pdf*

[2] Recommandations SUPANN, juillet 2003, http://www.cru.fr/_media/documentation/supann/supann-v10.pdf

[3] Security Assertion Markup Language, *OASIS Security Services*, http://www.oasis-open.org/committees/security/

[4] Shibboleth Enabled Applications and Services, Wiki Internet2, https://wiki.internet2.edu/confluence/display/seas/Home

[9] http://www.cru.fr/faq/federation/questions_reponses_pour_les_wayf#court-circuiter_le_wayf