Authentification multi plateforme dans un système d'information universitaire

Marc-Henri Boisis-Delavaud, Arunas Stockus, Jean-Marc Coris

Authentification multi plate-forme dans un système d'information universitaire

Marc Boisis-Delavaud
Centre de Ressources Informatiques
Université de La Rochelle
mdelavau@univ-lr.fr

Arunas Stockus
Centre de Ressources Informatiques
Université de La Rochelle
arunas.stockus@univ-lr.fr

Jean-Marc Coris
Consortium Cocktail
jean-marc.coris@cocktail.org

Résumé

En tant qu'éditeur de briques logicielles du système d'information donnant souvent accès à des informations sensibles et déployées dans des établissements aux architectures diverses, nous avons été contraints d'imaginer comment offrir pour ces applications un accès homogène (unifié) et présentant un bon niveau de sécurité.

C'est ce à quoi l'application ZAP tente de répondre.

Pour les utilisateurs, ZAP permet une « virtualisation » globale de l'accès aux applicatifs en présentant de manière homogène toutes les applications disponibles. En plus d'être un lanceur d'applications, ZAP offre des fonctionnalités comme l'authentification unique (SSO). ZAP s'appuie sur le serveur d'authentification CAS et propose une extension des fonctionnalités SSO aux applications « non Web » (exécutables natifs, applications Java). Elle intègre également les solutions permettant l'authentification utilisant les « tokens » USB RainBow iKey porteurs de certificats X509. L'application ZAP fonctionne sur les systèmes d'exploitation courants et permet de combiner plusieurs niveaux d'authentification (login/mot de passe, certificat).

Mots-clefs

Authentification, PKI, CAS, SSO, Java Web Start, Serveurs d'applications

1 Introduction

Le projet ZAP est né en voulant combler un certain nombre de besoins relatifs au système d'information. En effet l'université de La Rochelle est, de par les applicatifs de gestion qu'elle a développés en son sein, à l'origine du consortium « Cocktail ». Ce dernier comprend près d'une quarantaine d'établissements de l'enseignement supérieur qui utilisent et développent en commun un certain nombre d'applications utiles à leur fonctionnement. Il s'agit notamment d'applications de gestion financière ou de scolarité.

La diversité des architectures informatiques et des politiques de sécurité associées de ces différentes entités a conduit à une certaine industrialisation de nos méthodes de déploiement. Ces applications manipulant des données sensibles, nous nous devons d'offrir une sécurité d'accès au système d'information. En effet l'augmentation des utilisateurs nomades et la perte progressive de contrôle du poste de travail renforcent ce sentiment de vulnérabilité.

Nos utilisateurs accèdent à ces données sensibles grâce aux applicatifs, dont le nombre peut être assez élevé. La diversité des types d'applications rend leur lancement fastidieux pour les utilisateurs qui perdent beaucoup de temps à trouver le bon outil.

Notre but est donc de « virtualiser » l'accès aux applications, en les rendants accessibles simplement, quel que soit leur type. Lancer une application doit être aussi simple que le fait d'appuyer sur un bouton de télécommande. C'est justement cette idée qui était à l'origine de ce projet, et de son nom – « ZAP ».

Ainsi, ZAP est aussi un moyen simple de lancer toutes les applications. On ne s'authentifie qu'une seule fois et cela fonctionne sur tous les systèmes d'exploitation courants.

Dans cet article, nous allons énoncer les principes directeurs qui ont permis de mener à bien ce projet. Nous décrirons ensuite l'application ZAP et ses fonctionnalités. Enfin nous aborderons les points techniques permettant de mettre en place ces fonctionnalités et nous finirons par l'exposition des évolutions possibles de ce projet.

2 Principes directeurs

ZAP agit comme un frontal qui est le passage obligé pour lancer toutes les applications métiers, celles-ci donnant un accès en lecture, ajout, mise à jour ou suppression au niveau du système d'information.

ZAP est avant tout un lanceur d'applications qui permet de regrouper toutes les applications en un seul point d'accès pour l'utilisateur. A ce principe de base nous avons voulu ajouter un certain nombre de fonctionnalités.

Les applications recensées et publiées dans ZAP doivent l'être de façon dynamique afin de pouvoir effectuer une gestion centralisée et en temps réel des outils proposés. Ainsi, toute modification de l'ensemble des applications gérées par ZAP est immédiatement visible pour tous les utilisateurs.

Pour faciliter le lancement de multiples applications par l'utilisateur, nous avons souhaité mettre en place un système d'authentification unique pour toutes les applications, quelque soit leur mode de lancement ou d'accès.

Afin d'améliorer la sécurité, ZAP est en mesure de gérer plusieurs niveaux d'authentification, de l'accès anonyme jusqu'à l'authentification par certificat. Les certificats du CRU (Comité Réseau des Universités) étant déployés au sein de nos établissements, nous nous devions de développer un système capable d'exploiter cette possibilité d'authentification. De plus, ces derniers étant déposés dans des tokens cryptographiques, il nous fallait concevoir un système capable de dialoguer avec ces interfaces matérielles permettant une authentification forte.

Les environnements informatiques sont très variés d'un établissement à l'autre. Ainsi ZAP doit fournir une solution multi-plate-forme et multi-applications. Les systèmes d'exploitation supportés sont Microsoft Windows[1], MacOS X et Linux, ce qui permet de couvrir la quasi totalité des systèmes d'exploitation présents chez les utilisateurs. ZAP est également capable de lancer tous les types d'applications exploitées par nos utilisateurs : applications « Web », exécutables « natifs », etc.

3 Fonctionnement

ZAP est tout d'abord une application Java ayant une interface graphique. Bien qu'il puisse être considéré comme une application « indépendante », ces fonctionnalités sont au mieux exploitées lorsqu'il est intégré avec d'autres éléments de notre système. Notamment, le serveur ServAut [1] utilisé pour l'authentification des applications développées au sein du consortium Cocktail.

Ainsi, ZAP peut être vu comme une partie du système global d'authentification des utilisateurs et des applications.

3.1 Une interface graphique simple

L'interface graphique écrite en Java Swing est constituée de deux principaux éléments (Figure 1) :

[1] Nous emploierons le nom de « Windows » à la place de « Microsoft Windows » dans la suite de cet article pour simplifier la présentation.

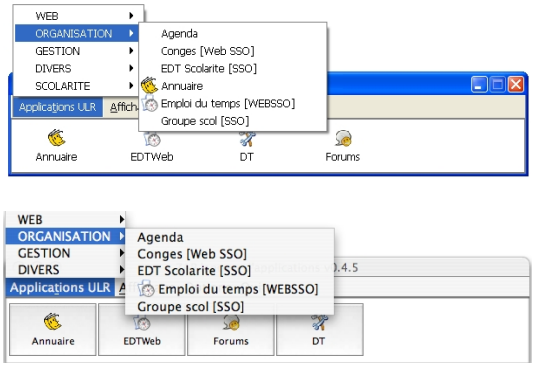

Figure 1 – *L'interface ZAP : le menu et la barre de lancement rapide. Interfaces Windows et MacOS X.*

– Un **menu** qui contient toutes les applications disponibles de l'établissement classées par thème. Les thèmes correspondent aux entrées du menu principal, et les applications sont accessibles dans les sous-menus. Le contenu de menu doit être établi par l'administrateur du système d'information.
– Une **barre de lancement rapide** contenant les applications favorites de l'utilisateur. Son contenu est libre pour que chacun puisse y déposer les outils les plus fréquemment utilisés. Il est constitué des entrées pouvant être prises dans le menu du ZAP.

Chaque application peut être symbolisée par une icône ce qui permet de la discerner plus rapidement. Elle peut être accompagnée d'un libellé ainsi que d'un texte explicatif.

Avec ce fonctionnement, l'accès à une application est complètement « virtualisée » : une application est une entrée dans le menu. ZAP permet à l'utilisateur de ne plus se soucier du type de l'application, ni de son emplacement.

3.2 Une description XML

Pour fonctionner, ZAP a besoin de connaître les applications disponibles pour le système d'exploitation dans lequel il est exécuté.

L'interface de ZAP, et notamment le menu de lancement des applications, est la représentation graphique d'une description au format XML contenant les applications avec leurs attributs (Figure 2).

Chaque balise `<application>` contient la description d'une application. Cette définition est composée d'attributs relatifs à la représentation de l'élément dans l'interface graphique : un nom, une icône, un libellé. D'autres attributs sont liés au mode de lancement de l'application : un type, une URL, un mode d'authentification.

Afin d'améliorer la présentation, les applications sont groupées par thème (la balise `<theme>`).

La Figure 2 présente un extrait d'un document XML avec la description de contenu pour ZAP. Les définitions de trois applications données dans cet exemple correspondent aux trois dernières entrées du menu ZAP affiché dans la Figure 1.

```
<applications
  name="Toutes les applications"
  xsi:noNamespaceSchemaLocation="AppliULR.xsd"
  comment="ZAP Application">

  <theme name="WEB" comment="Applications Web">
  ...
  </theme>

  <theme
    name="ORGANISATION" comment="Applications
d'organisation">
    ...
    <application  shortName="Annuaire"
      url="V:\Public\Apps\Annuaire.app\Annuaire.exe"
      name="Annuaire"
      authentication="login"
      type="ExeWindows"
      iconUrl="http://www.univ-
lr.fr/apps/zap/icones/annuaire32.png"
      comment="Annuaire de l'etablissement"/>
    <application  shortName="EDTWeb"
      url="http://www.univ-lr.fr/apps/edt"
      name="Emploi du temps [WEBSSO]"
      authentication="login" type="Web"
      iconUrl="http://www.univ-
lr.fr/apps/zap/icones/edt22.png"
      comment="Empoi du temps Web"/>
    <application  shortName="Groupe Scol"
      url="http://www.univ-lr.fr/cgi-
bin/WebObjects/GroupeScol.woa/1/eowebstart/com.webob
jects.eodistribution._EOWebStartAction/webStart/Java
Client.jnlp"
      name="Groupe scol [SSO]"
      authentication="certificat"
      type="WebStart"
      comment="Groupe scolarite"/>
  </theme>
...
</applications>
```

Figure 2 – *Un exemple de description XML pour ZAP.*

Lors de déploiement de ZAP dans un système d'information, la description des applications et leurs caractéristiques sont stockées en base de données. Afin de générer dynamiquement le catalogue des applications, un générateur, dont nous reparlerons plus loin (section 3.3), a été développé.

A son lancement, ZAP télécharge la liste des applications et construit, en fonction, son interface graphique. L'utilisateur peut à tout moment rafraîchir le contenu de ZAP, l'interface étant alors reconstruite.

Des filtres sont mis en place afin de ne proposer que les applications exécutables sur le système courant ; par exemple on ne va pas proposer une application Windows quand ZAP est exécuté sous MacOS X (Figure 1).

Le format XML a été choisi par soucis d'interopérabilité et pour la souplesse de ce langage. Il est donc facile de modifier les attributs afin de mieux qualifier les applications.

Comme nous l'avons vu, la barre de menu offre toutes les applications disponibles de l'établissement. La barre de lancement rapide est quant à elle le reflet d'une seconde description au format XML. Dans cette zone, l'utilisateur peut déposer ses applications favorites. Les deux descriptions XML (menu et barre de lancement) répondent au même schéma XML (un exemple de schéma XML est disponible sur le site de ZAP [2]).

La description XML relative à la barre de lancement étant propre à chaque utilisateur, elle est stockée dans un fichier enregistré dans le répertoire « home » de l'utilisateur. Conjointement à cette information, toute personnalisation des préférences ZAP est aussi stockée par l'utilisateur.

Il aurait été possible de stocker ces informations dans une source centralisée (ex., une base de données). Cela aurait permis à l'utilisateur de retrouver ses préférences tout en étant mobile. Cependant lors de la conception de ZAP, nous voulions une application « légère », sans contraintes liées à une dépendance d'une base de données (disponibilité d'un pilote JDBC, visibilité de la base en dehors du réseau d'établissement, etc.). On pourrait envisager l'utilisation de serveur ServAut [1] pour le stockage de ces préférences utilisateurs (voir la section 4).

3.3 Une source dynamique

Les fichiers XML ci-dessus représentent la source de ZAP et rassemblent l'ensemble des applications disponibles. Ces sources peuvent être « statiques », disponibles par exemple sous forme de fichiers XML sur un site Web. Elles peuvent également être générées dynamiquement (Figure 3).

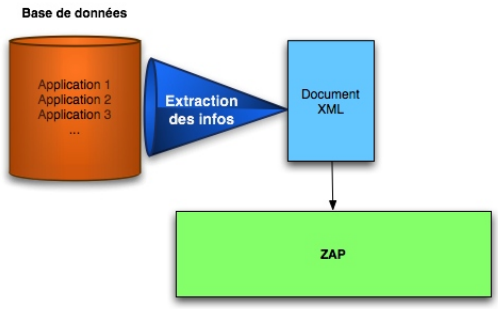

Figure 3 – *Initialisation dynamique de ZAP.*

C'est ce principe qui est utilisé dans le système d'information de l'université de La Rochelle. Une application spécifique, le serveur ServAut [1], est utilisée pour ces documents XML. ServAut est un serveur d'authentification, lui aussi développé à La Rochelle, permettant d'authentifier les applications et les modèles de données utilisés par celles-ci (voir la section 4 pour plus de détails).

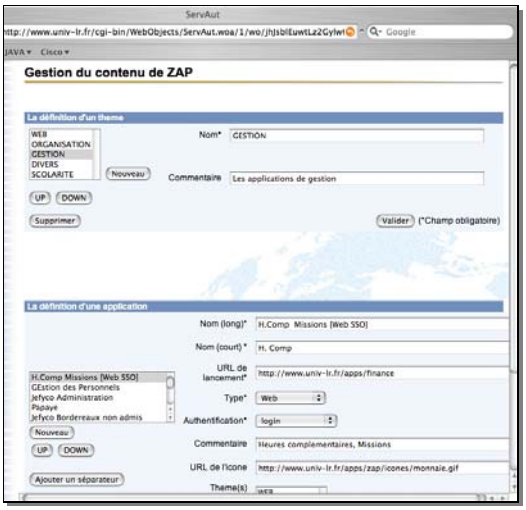

Figure 4 – *Interface d'administration du contenu de ZAP.*

Lors du déploiement, chaque développeur peut déposer une nouvelle application par le biais d'une interface Web intégrée dans le serveur ServAut (Figure 4). Le dépôt consiste à remplir un formulaire, où on spécifie le type d'application, un niveau d'authentification, une icône, et d'autres paramètres. Le formulaire validé, l'application est immédiatement disponible dans ZAP et peut donc être vue et utilisée par la communauté des utilisateurs.

3.4 Un Lanceur

3.4.1 Multi-plate-forme

ZAP est implémenté en Java et il peut être exécuté sur différents systèmes d'exploitation. Son fonctionnement a été plus particulièrement testé sur MacOS X, Linux et Windows.

Etant une application Java, ZAP lui-même peut être lancé en ligne de commande ou bien via Java Web Start (le principe de cette technologie est présenté ci-après).

Une version de Java 1.4 ou supérieure est nécessaire pour l'exécution de ZAP. Des bibliothèques Java externes sont aussi utilisées, dont les plus importantes sont :

- Apache Jakarta pour les communications chiffrées [7] ;
- IAIK pour l'interface PKCS#11 [9].

3.4.2 Multi-applications

ZAP doit être en mesure de lancer toutes les applications métier utilisées dans nos établissements. Dès sa conception, ZAP a été écrit de façon à rendre le code ouvert. Il serait donc simple d'implémenter un nouveau « lanceur » spécifique à un type d'application.

Aujourd'hui ZAP permet de lancer trois types d'applications : les exécutables natifs, les applications Web et les applications Java Web Start.

Exécutables natifs.

Il s'agit des applications qui peuvent être lancées en ligne de commande : un exécutable binaire compilé pour un système d'exploitation cible, un script de commandes « shell » ou encore une application Java.

Le principe de lancement des exécutables natifs est relativement simple : ZAP utilise les fonctionnalités de la classe Java `Runtime` permettant d'exécuter une commande avec arguments dans un processus séparé de la machine virtuelle Java (JVM).

Applications Web.

Il s'agit d'applications qui sont installées et exécutées sur un serveur d'applications et qui sont accessibles par le biais d'un navigateur Web. ZAP intègre une solution permettant de lancer un navigateur Web du système client avec une URL passée en paramètre.

Applications Java Web Start.

La technologie Java Web Start est utilisée pour déployer les applications Java via le réseau [3]. En quelques clics, une application Java (l'ensemble des ces bibliothèques Java JAR) peut être téléchargée et exécutée sur le poste utilisateur.

ZAP lance les applications Java Web Start grâce à l'outil `javaws` fourni avec les installations Java. `javaws` se connecte à un serveur Web à partir duquel il télécharge un fichier de description JNLP (Java Network Launching Protocol). Ce fichier contient notamment les adresses des bibliothèques nécessaires au lancement de l'application et de ses paramètres de démarrage. Nous invoquons donc l'exécutable `javaws` en lui passant en paramètre l'URL du fichier JNLP de l'application souhaitée.

A noter que les applications Web et Java Web Start constituent l'essentiel des applications développées et distribuées par le consortium Cocktail. Elles sont développées et déployées sur le serveur d'applications WebObjects d'Apple.

En plus du lancement des applications, ZAP propose des solutions pour l'authentification unique des utilisateurs.

3.5 Single Sign-On

Le Single Sign-On, noté SSO, permet de centraliser l'authentification des utilisateurs sur des applications pour lesquelles ils sont autorisés, avec pour objectif de propager l'information d'authentification à ces différentes applications et d'éviter ainsi à l'utilisateur de multiples identifications par mot de passe.

ZAP assure cette fonctionnalité SSO, c'est-à-dire que l'utilisateur ne s'authentifie qu'une fois et peut ensuite lancer une multitude d'applications sans se ré-authentifier.

3.5.1 Solution basée sur CAS

CAS (Central Authentication Service) met en oeuvre un serveur d'authentification accessible via Web. Initialement développé à l'université de Yale, CAS fait désormais partie des projets de JA-SIG [4]. Le projet ESUP-Portail, dédié au développement des espaces numériques de travail (ENT), propose des extensions de CAS facilitant son déploiement dans les systèmes d'information et son intégration avec des sources d'authentification variées (serveurs LDAP, bases de données, etc.) [5]. CAS est le principal serveur d'authentification pour les applications intégrées dans cet ENT [6].

CAS étant fiable et largement utilisé dans les systèmes d'information universitaires, nous avons voulu étendre ses fonctionnalités à tout type d'applications. En matière de SSO, ZAP peut être vue comme une extension de CAS qui s'appuie complètement sur ce dernier.

En termes d'authentification, ZAP est un intermédiaire entre les applications et CAS. Son fonctionnement reprend exactement les mêmes principes qu'un navigateur Web lorsqu'il est utilisé pour l'authentification des applications Web auprès de CAS (Figure 5).

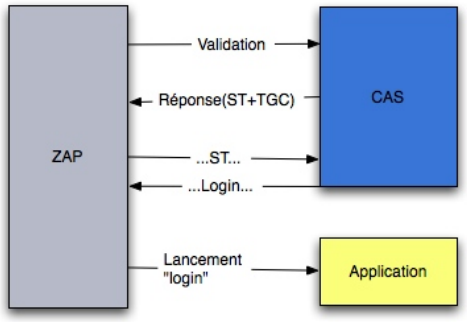

Figure 5 – *Lancement d'une application Web en utilisant l'authentification CAS.*

Lors du premier lancement d'une application quelconque nécessitant une authentification par login et mot de passe, ZAP interroge l'utilisateur pour connaître ses identifiants comme pourrait le faire un formulaire du serveur CAS affiché dans un navigateur Web. Ces identifiants sont ensuite transmis au serveur CAS pour validation. S'ils sont validés par CAS alors ZAP reçoit en réponse un ticket ST (Service Ticket). La réponse HTTP inclut aussi un cookie appelé TGC (Ticket Granting Cookie). Le ticket ST, utilisable une seule fois, permet à ZAP de recevoir l'identifiant d'utilisateur validé.

La deuxième connexion au serveur CAS utilisant le ticket ST n'est pas réellement nécessaire, car ZAP connais déjà l'identité de l'utilisateur validé. Elle est effectuée afin de « consommer » ce ticket dans le CAS.

Le cookie TGC, mémorisé par ZAP et ajouté aux futures requêtes envoyées au CAS, permet de ré-authentifier l'utilisateur auprès de CAS sans re-saisir ses identifiants lors du lancement d'autres applications.

La dernière étape consiste à lancer l'application et de lui transmettre le login d'utilisateur. L'application peut ainsi commencer directement ses tâches personnalisées.

Les communications entre CAS et ZAP sont faites en HTTPS, protocole chiffré, afin de ne pas faire transiter les identifiants de manière lisible. Pour s'assurer de l'authenticité du serveur nous validons son certificat. Ces implémentations de client HTTPS ont été rendues possibles grâce à l'utilisation des bibliothèques *HttpClient* de projet Apache Jakarta [7]. Les certificats racines sont embarqués dans ZAP.

Dans un souci de sécurité, nous avons décidé de revalider l'authentification de l'utilisateur auprès de CAS au lancement de chaque application. Ainsi l'utilisateur a l'impression de s'authentifier seulement une fois mais en réalité CAS revalide le ticket fournit lors de la première authentification. Cela permet de placer CAS comme seul maître de l'authentification et il est averti de toute authentification. Le coût en temps de cette ré-authentification, transparente pour l'utilisateur, est négligeable.

3.5.2 Lancement d'applications

Nous avons vu les communications entre CAS et ZAP, nous allons maintenant étudier la transmission de l'identifiant entre ZAP et l'application lancée. Cette identifiant est nécessaire pour la plupart des applications métier dont le comportement dépend du profil de l'utilisateur.

Le lancement des applications a déjà été présenté dans la section 3.4.2. Le principe de transmission d'identifiant d'utilisateur est lui aussi inspiré du fonctionnement de CAS. Similaire à CAS, ZAP utilise un protocole de communication avec les applications employant les tickets ST et TGC. Le mécanisme de lancement des applications varie en fonction de leur type : exécutables, Java Web Start ou applications Web. Afin d'éviter qu'une personne malintentionnée puisse lancer une application en spécifiant le login de l'utilisateur, nous utilisons un système de callback permettant de s'assurer que le login provient de ZAP.

Exécutables natifs.

Le protocole de communication entre ZAP et les applications lancées en tant qu'exécutables natifs est basé sur le principe du callback (Figure 6). Le dialogue est initié par l'application qui contacte ZAP juste après s'être lancée.

Lors de lancement d'une application, ZAP génère et lui fournit un ticket, équivalent de Service Ticket de CAS, ainsi qu'un numéro de port. Ils correspondent respectivement aux paramètres `LRAppDockTicket` et `LRAppDockPort` passés en ligne de commande. Par exemple, une ligne de commande suivante peut être utilisée pour lancer une application :

```
application.exe \
    -LRAppDockTicket 625452765826841672 \
    -LRAppDockPort 61134
```

Après avoir lancé l'application, ZAP se place en attente de connexion sur le port indiqué. Le numéro de port est tiré aléatoirement au lancement de ZAP afin de permettre l'utilisation de plusieurs instances de ZAP sur la même machine. L'application extrait ce ticket et ce numéro de port. Ensuite elle se connecte à ZAP sur le port donné et envoi son ticket. ZAP vérifie le ticket et s'il est correct, il retourne l'identifiant de l'utilisateur en guise de réponse. Le ticket ne peut être utilisé qu'une seule fois.

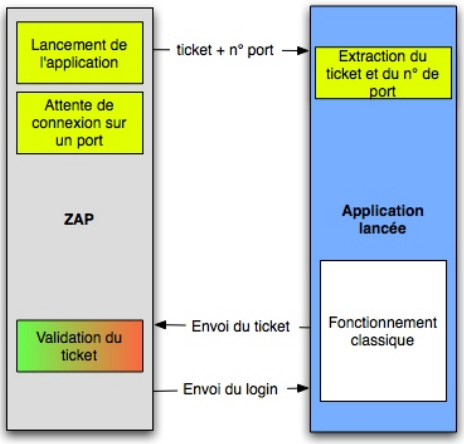

Figure 6 – *Lancement et authentification pour les applications « exécutbles ».*

Il est prévu que si un problème intervient lors de communication entre ZAP et une application, celle-ci doit utiliser une authentification classique, un panneau de login par exemple. Ainsi chaque application reste autonome et peut toujours fonctionner sans ZAP.

Applications Java Web Start.

Le lancement des applications Java Web Start suit le même fonctionnement que les exécutables natifs. Un ticket et un numéro de port sont transmis à l'application qui ensuite doit re-contacter ZAP pour récupérer l'identifiant de l'utilisateur. La différence consiste dans la manière de passer les paramètres à l'application.

Comme nous l'avons déjà mentionné, ces applications sont lancées grâce à un fichier de description JNLP. ZAP télécharge donc le fichier JNLP et le modifie pour y ajouter les paramètres correspondant au ticket et au numéro de port. Il utilise ensuite l'utilitaire `javaws` pour lancer l'application.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://www.univ-
lr.fr:80/WebObjects/Java">
  <information>
    <title>GEDI</title>
    <homepage href="http://www.univ-lr.fr:80/cgi-
bin/WebObjects/GEDI2.woa"/>
    ...
  </information>
  <security><all-permissions/></security>
  <resources>
    <j2se version="1.4+"/>
    <jar download="eager" href="ClientSTD.jar"/>
    <jar download="eager" href="GEDI2.jar"/>
     ...
  </resources>
  <application-desc main-
class="com.webobjects.eoapplication.client.EOClientA
pplicationSupport">
    <argument>-applicationURL</argument>
    <argument>http://www.univ-lr.fr:80/cgi-
bin/WebObjects/GEDI2.woa</argument>
    <argument>-page</argument>
    <argument>JavaClient</argument>
    <argument>-suppressClassLoading</argument>
    <argument>true</argument>
  </application-desc>
</jnlp>
```

Figure 7 – *Le fichier JNLP original utilisé pour le lancement d'une application en mode Java Web Start.*

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://www.univ-
lr.fr:80/WebObjects/Java">
  ...
  <application-desc main-
class="com.webobjects.eoapplication.client.EOClientA
pplicationSupport">
    <argument>-LRAppDockTicket</argument>
    <argument>625452765826841672</argument>
    <argument>-LRAppDockPort</argument>
    <argument>61134</argument>
    <argument>-applicationURL</argument>
    <argument>http://www.univ-lr.fr:80/cgi-
bin/WebObjects/GEDI2.woa</argument>
    <argument>-page</argument>
    <argument>JavaClient</argument>
    <argument>-suppressClassLoading</argument>
    <argument>true</argument>
  </application-desc>
</jnlp>
```

Figure 8 – *Le fichier JNLP modifié par ZAP pour le lancement d'une application en mode Java We Start.*

Un extrait d'un fichier JNLP « original » permettant de lancer une application JavaClient de WebObjects est donné dans la Figure 7.

La Figure 8 contient un extrait du fichier modifié par ZAP (dans cet exemple, uniquement la partie entre les balises `<application-desc>` est laissée). La modification consiste donc à ajouter les balises `<argument>` dans la section `<application-desc>` sans modifier d'autres éléments du fichier.

Applications Web.

L'accès aux applications Web utilise un navigateur Web. Il est donc nécessaire de lui transmettre les informations qui permettraient au serveur CAS d'authentifier un utilisateur lorsque ceci sera demandé par l'application. ZAP le fait en ajoutant le cookie TGC dans la mémoire de navigateur.

Plus précisément, ZAP lance le navigateur Web en spécifiant une URL à ouvrir (Figure 9). L'URL contient l'adresse de la machine locale et le numéro de port. Elle demande l'ouverture d'un « document » qui en effet correspond au ticket ZAP (voir précédemment). Par exemple, un navigateur Web peut être lancé par ZAP avec l'URL suivante:

```
http://ma.machine.domaine:61134/625452765826841672
```

Le navigateur envoie donc une requête HTTP à l'URL indiquée. ZAP extrait le ticket de la requête, le valide et identifie l'application qui doit être lancée. Il construit ensuite une réponse HTTP qui redirige le navigateur vers la page d'accueil de l'application. Cette réponse ajoute également le cookie TGC comme le cookie de session du navigateur (en effet, il s'agit d'une copie du cookie délivré par le serveur CAS).

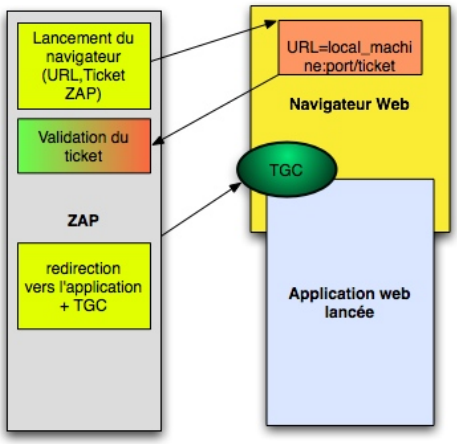

Figure 9 – *Lancement des applications « Web » via ZAP.*

Pour l'authentification des utilisateurs, l'application Web fonctionne ensuite de manière habituelle – elle accède au serveur CAS. Le cookie TGC étant présent dans le navigateur Web, un utilisateur peut être authentifié par CAS sans avoir besoin de re-saisir son login et mot de passe. D'autre part, si après la validation il s'avère que le cookie est périmé ou non reconnu, le serveur propose le formulaire d'authentification accessible via une connexion sécurisée et un nouveau cookie TGC est ajouté dans le navigateur.

Cette solution a l'avantage d'être utilisable avec toutes les applications Web supportant l'authentification CAS. Aucune modification dans le code des applications n'est nécessaire pour pouvoir les lancées via ZAP. Mais elle a également une limitation : le cookie TGC est pris en compte seulement si ZAP et CAS sont exécutés sur le même domaine. Le cookie TGC ne sera pas reconnu par CAS si ZAP est exécuté, par exemple, en dehors du réseau de l'établissement.

3.5.3 Proxy CAS ?

Nous avons étudié le mécanisme de proxy CAS, dédié à l'authentification CAS pour les applications « non Web ». Dans ce cas, ZAP peut utiliser un Proxy Granting Ticket (PGT) délivré par le serveur CAS. L'application cliente reçoit ensuite un Proxy Ticket (PT) qu'elle valide directement auprès de CAS. Ce PT s'obtient grâce à une communication chiffrée entre CAS et ZAP, celui-ci devant accepter les connexions HTTPS.

Ceci pose un problème conceptuel en matière de certificat électronique. En effet ZAP est une application cliente qui peut s'exécuter sur une multitude de postes, qui dans ce cas se comporteraient comme autant de serveurs HTTPS. Il apparaît évident que nous ne pouvons pas délivrer de certificats serveur à chaque machine pouvant exécuter ZAP.

L'utilisation de mécanisme proxy CAS reste pour nous un objectif car il serait synonyme de compatibilité totale avec le mode de fonctionnement CAS.

3.6 L'authentification

Le niveau d'authentification est spécifié pour chaque application. Il correspond à l'attribut `authentication` de la balise `<application>` dans la description XML pour ZAP (voir la section 3.2).

ZAP propose trois niveaux d'authentification :

- Nulle, pour les applications ne nécessitant pas d'authentification (valeur `"none"` de l'attribut `authentication`).
- Authentification par login et mot de passe (valeur `"login"`).
- Authentification par certificat et code PIN (valeur `"certificat"`).

Les deux derniers niveaux peuvent être combinés afin d'assurer une sécurité maximale. Le lancement de l'application nécessite dans ce cas l'authentification par le login et le mot de passe, mais aussi par certificat (valeur `"login+certificat"`).

3.6.1 Login et mot de passe

Comme nous l'avons présenté précédemment, l'authentification par login et mot de passe s'appuie complètement sur le serveur d'authentification CAS (section 3.5).

3.6.2 Certificat et token cryptographique

Des tokens de sécurité USB (Rainbow iKey 3000) sont distribués aux utilisateurs. Ils contiennent un certificat personnel délivré par l'autorité de certification du CRU. L'authentification forte permet de sécuriser des applications qui manipulent les données sensibles comme celles de la gestion financière, par exemple.

Ces tokens respectent la norme PKCS#11 [8] ce qui permet de les accéder à partir des applications. En java, nous avons utilisé la bibliothèque IAIK [9] permettant de communiquer avec le token et de lire son certificat.

Lors du lancement d'une application exigeant une authentification par certificat, l'utilisateur renseigne son code pin puis ZAP lit le certificat X509. Un certain nombre d'attributs du certificat sont vérifiés : la validité, le certificat racine, la non révocation, le code de l'établissement. Les listes de révocations (CRL) sont téléchargées lors de chaque validation.

Si l'application lancée a besoin d'un login pour identifier l'utilisateur et que ce login n'est pas présent dans le certificat, ZAP effectue une recherche dans l'annuaire LDAP afin de le retrouver. La recherche se fait généralement sur la clé publique de la personne.

Cette authentification par certificat seulement ne permet pas d'avoir une fonctionnalité SSO pour les applications Web. Nous avons vu que les applications Web nécessitent un cookie CAS, or CAS ne gère que l'authentification par login et mot de passe, ce dernier n'étant pas connu lors de la lecture du certificat. La solution consiste, pour les applications Web sensibles, d'utiliser une double authentification : login / mot de passe et certificat.

La société Rainbow fournit des pilotes des tokens pour les systèmes Windows et Linux. La bibliothèque IAIK permet de communiquer avec ces pilotes afin de pouvoir accéder aux certificats électroniques. Grâce à cette bibliothèque, ZAP peut aussi utiliser d'autres tokens respectant la norme PKCS#11 sachant que ce protocole se place au dessus des pilotes.

L'utilisation des tokens sous MacOS X est plus problématique. En effet la société Rainbow fournit des pilotes en version non-officielle dont la stabilité n'est pas validée. Avec ce système d'exploitation, la meilleure solution serait de pouvoir utiliser l'application native de MacOS X `keychain.app`. Cette application est un trousseau rassemblant tous les identifiants et le certificat de l'utilisateur. Malheureusement il n'existe pas, actuellement, d'interface pour l'utiliser à partir des applications Java.

Pour conclure sur l'utilisation des tokens cryptographiques, nous pouvons affirmer qu'ils apportent un plus en matière de sécurité. En effet cela permet de garantir un accès fortement sécurisé à certains applicatifs sensibles. D'autre part, l'interface matérielle induit des contraintes relatives aux systèmes d'exploitation, cela a pour effet de réduire la portabilité de ZAP. Des solutions sont à étudier quant à la virtualisation de ces tokens, notamment des projets de serveurs de PKI (Public Key Infrastructure) [10].

4 Serveur d'authentification ServAut

Nous avons mentionné que ZAP peut être utilisé conjointement avec le serveur d'authentification ServAut. ServAut fournit une interface Web permettant de définir le contenu de ZAP (voir la section 3.3). Il permet également de générer dynamiquement une description en format XML pouvant être exploitée par ZAP.

Le serveur ServAut lui-même est une application WebObjects écrite en Java et pouvant être déployée sur ce serveur d'applications [11]. On utilise la technique dite *DirectActions* permettant grâce à une URL spécialement formatée d'exécuter les méthodes de classes Java de l'application et d'obtenir le résultat de leur exécution sous forme d'un document quelconque.

Dans le cas d'une description XML pour ZAP, l'URL suivante doit être utilisée (présentée sur plusieurs lignes) :

```
http://mon.serveur.applis/cgi-bin/WebObjects/
      ServAut.woa/wa/getXMLZAP
```

Cette URL indique que la méthode `getXMLZAPAction` de la classe `DirectAction` (la classe par défaut des « actions » dans une application WebObjects) doit être exécutée. Cette méthode génère et retourne une description XML des applications pour ZAP.

Les fonctionnalités de ServAut ne se limitent pas au ZAP. Il est également utilisé pour *l'authentification des utilisateurs* à partir de leur login et mot de passe. Avec la généralisation d'utilisation de serveurs CAS dans nos établissements, cette fonctionnalité devient obsolète et elle est de moins en moins utilisée.

L'authentification des applications est la fonctionnalité la plus importante du serveur ServAut, celle pour laquelle il a été initialement créé.

Les applications du consortium Cocktail intègrent toutes des modèles de données définissant une mise en correspondance entre la représentation des données dans la

mémoire de l'application et celle dans une source de données (SGBDR le plus souvent). Elles ont donc toutes besoin de disposer d'informations appelées « dictionnaire de connexion » : l'URL JDBC de connexion à une base de données, le nom d'utilisateur, son mot de passe.

Le serveur ServAut gère une liste de tous les dictionnaires pouvant être utilisés par les applications. Au moment de leur lancement, ce serveur permet de communiquer un dictionnaire de connexion pour chaque modèle de données utilisé par une application. L'échange d'informations entre les applications et ServAut s'effectue via une communication chiffrée.

Une intégration plus étroite avec ZAP est prévue dans les futures versions de ServAut. Nous envisageons de l'intégrer dans une chaîne d'authentification CAS et ZAP, et notamment celle utilisant le mécanisme de proxy CAS. Nous étudions également une solution où le niveau d'authentification minimum autorisé pourrait être fixé pour les applications (login/mot de passe, certificat ou les deux). ServAut devra alors être utilisé comme le serveur de la validation de ce niveau, empêchant son redéfinition par un utilisateur non autorisé.

5 Développement pour ZAP

Les applications Java et exécutables natifs doivent être adaptés pour pouvoir s'intégrer dans le système d'authentification proposé par ZAP :

- Prendre en compte les paramètres ZAP passés en ligne de commande (`LRAppDockTicket` et `LRAppDockPort`).
- Pouvoir établir une connexion avec ZAP au port indiqué et récupérer le login d'utilisateur authentifié.

Si l'application n'est pas lancée via ZAP (alors les paramètres ne sont pas disponibles), ou si la communication avec ZAP échoue, elle doit être capable de procéder à une authentification personnelle en proposant un panel de login et de mot de passe, par exemple. Ce dernier point assure l'indépendance des applications vis-à-vis de ZAP.

Pour les applications Web, elles doivent supporter l'authentification par CAS, quelque soit leur langage et l'environnement de développement ou de déploiement. Aucune autre modification n'est nécessaire pour qu'elles puissent être lancées et authentifiées via ZAP.

Les applications développées au sein de consortium Cocktail sont pour la plupart implémentées en Java et utilisent des bibliothèques communes (« frameworks » dans la terminologie de WebObjects) [12]. Ces bibliothèques offrent des solutions pour le support de l'authentification CAS dans les applications Web. Elles contiennent également des classes facilitant la communication avec ZAP à partir des applications Java.

Le framework *CRIWebApp* contient la classe `LRDockClient` implémentant la communication avec le ZAP. Sa méthode statique `getNetID` permet d'établir une connexion avec le ZAP et de récupérer le login de l'utilisateur correspondant au ticket donné. Un exemple du code Java est donné dans la Figure 10.

Pour les applications Web, le framework CRIWebApp propose des actions permettant de gérer l'authentification CAS. Pour plus de détails, on peut se référer à la classe `CRIWebAction` proposant les méthodes de communication avec le serveur CAS et celles de la gestion de « callback ».

```
public String getUserName() {
  // On recupere les parametres passes en ligne
  // de commande
  String zapPort =
      commandLine().get("LRAppDockPort");
  String zapTicket =
      commandLine().get("LRAppDockTicket");
  String userName = null;

  // Les deux parametres doivent etre connus pour
  // communiquer avec ZAP. On suppose qu'il est
  // disponible sur le poste local
  //      getNetID(null, ...)
  if ((zapTicket != null) && (zapPort != null))
    userName =
     LRDockClient.getNetID(null, zapPort, zapTicket);
  if (userName == null) {
    // proposer un panel de login et de mot de passe
    // propre a l'application afin d'authentifier
    // l'utilisateur localement
    // ...
  }
  return userName;
}
```

Mis en forme : Français France

Figure 10 – *Exemple de code Java pour la communication avec ZAP.*

Les bibliothèques citées contiennent les implémentations plus spécifiquement dédiées aux applications WebObjects. Certaines d'entre elles peuvent tout de même être considérées comme étant « pure Java » et utilisées dans tout type d'applications Java.

6 Conclusions et évolutions

L'application ZAP désormais distribuée au sein du consortium Cocktail montre déjà son intérêt notamment dans les établissements à la problématique de déploiement complexe et multi-plate-forme côté client.

Pour les administrateurs d'un système d'information, ZAP apporte des fonctionnalités visant à simplifier leur travail. La publication des applications est facile et centralisée au niveau de l'application ServAut. Elle permet de déclarer l'ensemble des applications disponibles aux utilisateurs, leurs modes de lancement et les niveaux d'authentification requis. Cette information étant lue par ZAP en temps réel (au démarrage ou sur demande de l'utilisateur), tous les utilisateurs de ZAP peuvent ainsi être immédiatement informés de toute modification survenue dans les définitions des applications.

Pour les utilisateurs, ZAP simplifie le lancement des applications sans devoir se soucier de l'emplacement de l'application, ni de son mode de lancement. La fonctionnalité de l'authentification unique, quelque soit le type de l'application, est également très appréciable. Enfin, ZAP fournit aux utilisateurs une vue homogène sur l'ensemble des applications disponibles dans l'établissement.

ZAP c'est aussi son système d'authentification unique (SSO) permettant à l'utilisateur d'être authentifié une seule fois pour le pool d'applications mis à sa disposition. Ce SSO se base sur des technologies existantes comme le serveur d'authentification CAS mais en plus, il intègre des solutions permettant l'authentification d'utilisateurs par certificat (et le code PIN). C'est d'ailleurs, il faut le rappeler, cette dernière fonctionnalité *(authentification forte à base de certificats X509*) qui a été à l'origine de la création de ZAP.

Des évolutions sont envisagées dans les futures versions Elles concernent l'interface graphique de l'application perfectible sur la plate-forme MacOS X mais aussi une meilleure gestion des certificats utilisateurs sur cette plate-forme. L'intégration dans ZAP de support des « proxy CAS », une technique permettant l'authentification CAS des applications non-Web, est également à l'étude.

Bibliographie

[1] ServAut – Serveur d'authentification des applications et d'utilisateurs. *http://www.univ-lr.fr/apps/support/servaut/*

[2] ZAP – un lanceur d'applications. *http://www.univ-lr.fr/apps/support/zap/*

[3] Java Web Start Technologie. *http://java.sun.com/products/javawebstart/*

[4] CAS – Central Authentication Service. *http://jasigch.princeton.edu:9000/display/CAS*

[5] ESUP-Portail. *http://esup-portail.org*

[6] P. Aubry, V. Mathieu et J. Marchal. ESUP-Portail: open source Single Sign-On with CAS (Central Authentication Service). *10th International Conference of European University Information Systems - EUNIS 2004*, Bled (Slovenie), Juin 2004.

[7] Bibliothèques Apache Commons-HttpClient, projet Jakarta : *http://jakarta.apache.org/commons/*

[8] PKCS #11: Cryptographic Token Interface Standard. *http://www.rsasecurity.com/rsalabs/node.asp?id=2133*

[9] IAIK - Bibliothèque Java implémentant pkcs#11 : *http://jce.iaik.tugraz.at/products/*

[10] W. Schneider, Client-Server-Based Model for PKI-Services. *TERENA networking conference 2004*, Rhodes (Grece), Juin 2004.

[11] Apple WebObjects. *http://developer.apple.com/webobjects/*

[12] CRIWebApp – une bibliothèque des extensions WebObjects pour les applications Cocktail. *http://www.univ-lr.fr/apps/support/common/*