Annuaire LDAP, SSO et certificats du CRU à l'Université Paris 1

Fabrice Jammes
CRIR, Université Paris 1
Fabrice.Jammes@univ-paris1.fr

Benoît Branciard
CRIR, Université Paris 1
Benoit.Branciard@univ-paris1.fr

David Chopard-Lallier
CRIR, Université Paris 1
David.Chopard-Lallier@univ-paris1.fr

Yvonne Girard
CRIR, Université Paris 1
Yvonne.Girard@univ-paris1.fr

Résumé

Le service d'authentification SSO est un point stratégique du système d'information. Optimiser la sécurité, la continuité et la qualité de ce service est donc crucial. Dans cette optique, nous vous présentons trois techniques majeures :

- La mise en place d'une réplication LDAP sécurisée par SSL/TLS, utilisant les certificats du CRU, avec un chiffrement de l'information et une authentification mutuelle des parties.

- L'optimisation de la base de données BDB, utilisée par LDAP, pour la gestion d'un nombre d'utilisateurs importants.

- Un outil de reprise sur panne du service de SSO : cas-spare. Cet outil est efficace, très simple à installer et à configurer. Il est en production à l'Université Panthéon-Sorbonne et est disponible sous license LGPL. Nous montrons comment cas-spare assure, avec un effort minimum de votre part, la robustesse de votre service d'authentification.

Mots clefs

authentification, SSO, CAS, LDAP, réplication, certificats X509, SSL, TLS, logiciels libres

1 L'authentification, clé de voute du système d'information

Les systèmes d'informations actuels s'orientent vers la personnalisation de l'information et une gestion intelligente du profil des utilisateurs. Chaque personne doit être reconnue et donc authentifiée au sein de l'environnement numérique de travail [1].

Ainsi, la stabilité du service d'authentification est un prérequis au bon fonctionnement de l'ENT. Quels sont les maillons faibles de ce service ?

Pour la grande majorité des universités ce service est composé :
– d'une base de données contenant l'identifiant et le mot de passe des utilisateurs. Le plus souvent cette base de données est un annuaire LDAP conforme à la norme SUPANN ;
– d'un service de Single-Sign-On interrogeant l'annuaire, le plus souvent le serveur CAS.

Cet article propose des solutions permettant :
– d'optimiser la robustesse du service LDAP grâce à une réplication sécurisée et un paramétrage adéquat de la base de données ;
– de reprendre les pannes du service CAS avec souplesse et simplicité, grâce à l'utilitaire *cas-spare*.

2 LDAP à l'Université Panthéon-Sorbonne : état des lieux

L'Université Paris 1, implantée sur 25 sites géographiques à Paris et en banlieue, est une des plus grandes de France avec plus de 40000 étudiants et 350 diplômes. Elle dispose d'un annuaire d'environ 60000 comptes, et de deux réplicats, sur lesquels une sauvegarde journalière est effectuée.

Ces trois annuaires constituent le référentiel du système d'information de l'université :
– L'accès à internet dans les salles de libre-service est ouvert aux personnes authentifiées par un serveur de proxy-mité Squid. L'accès au réseau sans-fil ainsi que certains accès au réseau filaire sont quant à eux gérés par un serveur freeRadius. Ces deux services d'authentification s'appuient sur nos annuaires LDAP.
– De plus, ces annuaires permettent d'assurer, avec le service de SSO CAS, l'authentification des utilisateurs auprès de l'ensemble des services de l'environnement numérique de travail.

Notre configuration optimale d'OpenLDAP nous permet d'assurer un service d'annuaire avec des temps de réponses excellents. De plus, notre système de reprise sur erreur de CAS, *cas-spare*, minimise les temps d'indisponibilité en cas de défaillance du serveur SSO. Néanmoins, notre service d'authentification n'a pas encore atteint sa charge maximale, mais il fonctionne sur une architecture matérielle peu puissante (des serveurs PIII-600 avec 650 Mo de RAM) et nous sommes confiants quant à ses performances futures.

3 Réplication sécurisée d'annuaire LDAP

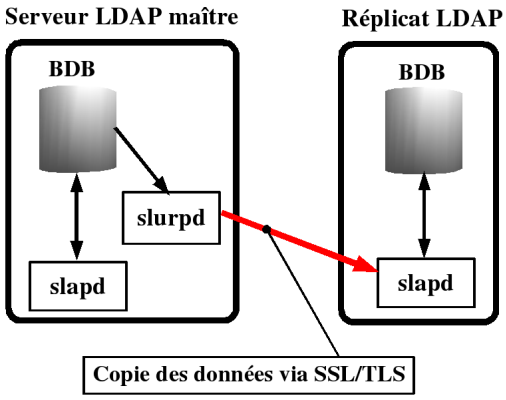

Figure 1 – *Processus de réplication avec OpenLDAP*

Une réplication sécurisée grâce aux certificats X509 permet de s'assurer de l'authenticité des serveurs LDAP maître et esclaves. Ainsi on interdira à un *faux* maître de se répliquer sur un esclave authentique, et à un maître authentique de se répliquer sur un *faux* esclave. Le chiffrement des données transmises sera également assurée.

3.1 Mécanisme de réplication LDAP

La technique de réplication mise en place est classique [2], elle est décrite sur la figure 1. Le démon slapd s'exécute sous le compte système slapd sur le maître et ses réplicats. Le démon slurpd s'exécute sous le compte root sur le maître seulement.

Les certificats de l'ac-serveur du CRU ont été installés sur le serveur LDAP ainsi que sur les réplicats.

On peut noter que lors de la réplication, le processus slurpd s'exécutant sur le maître va se connecter en tant que client LDAP sur les esclaves afin de recopier les données de l'annuaire.

3.2 Mécanismes SSL/TLS dans OpenLDAP

Le tutorial de Jean-Luc Archimbaud est une parfaite introduction aux problématiques de certificats électroniques [3].

Pour une compréhension plus avancée, des documentations complètes sont disponibles sur le site du CRU [4], notamment le tutoriel IGC présenté au JRES2001 [5].

Voici les mécanismes de communication entre un serveur OpenLDAP et un client LDAP via une connexion SSL/TLS avec des certificats :

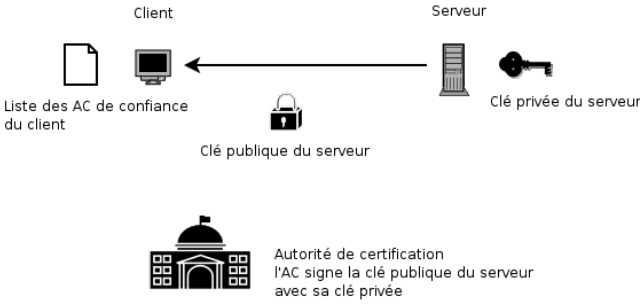

Figure 2 – *Authentification du serveur par le client avec SSL*

La procédure décrite dans la figure 2 permet au client d'authentifier le serveur, et de valider que la clé publique du serveur a bien été signée par l'IGC du CRU.

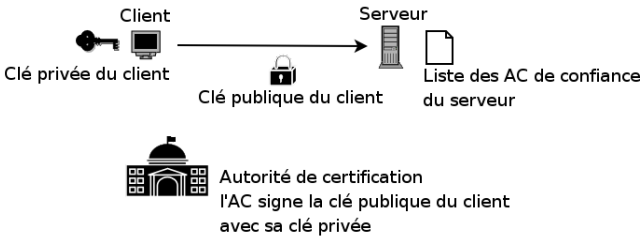

Figure 3 – *Authentification du client par le serveur avec SSL*

La procédure décrite dans la figure 3 permet au serveur d'authentifier le client, et de valider que la clé publique du client a bien été signée par l'IGC du CRU. Cette procédure doit être rendue obligatoire afin de sécuriser complètement le processus de réplication.

Le maître OpenLDAP doit accepter des connexions SSL/TLS de clients LDAP, et doit donc disposer d'un **certificat avec une extension TLS serveur**. De plus, le démon slurpd initie une session TLS quand il se connecte aux réplicats. Il doit alors utiliser un certificat avec une **extension TLS cliente**. Ce certificat est lui aussi émis par l'**ac-serveur** du CRU qui délivre les certificats de service (par opposition aux certificats de personnes). Dans notre cas, on utilise un seul certificat contenant les deux extensions.

3.3 Configuration TLS d'un serveur OpenLDAP

Une documentation précise [6] est disponible dans la FAQ du site du CRU. On va configurer[1] le serveur OpenLDAP pour qu'il utilise les certificats du CRU, et qu'il accepte uniquement des connexions TLS de clients possédant des certificats du CRU.

Dans /etc/ldap/slapd.conf :

♯ Configuration de l'encryption TLS :
TLSCipherSuite HIGH :MEDIUM :+SSLv2
♯ Fichier contenant les certificats de toutes les autorités
♯ de certification auxquelles on fait confiance.
♯ Ce fichier est la liste de toute les AC de confiance
♯ du serveur.
TLSCACertificateFile /etc/ssl/certs/ac–bundle.pem
♯ Clé publique délivrée par le CRU, en accès libre
TLSCertificateFile /etc/ssl/certs/server.crt
♯ Clé privée délivrée par le CRU, accessible en lecture
♯ seule au compte slapd
TLSCertificateKeyFile /etc/ssl/private/server.key
♯ La vérification de l'authenticité du
♯ client est obligatoire.
TLSVerifyClient demand

3.4 Configuration TLS d'un client LDAP

On indique que le client peut se connecter en TLS uniquement à des serveurs possédant un certificat du CRU, dans le fichier /etc/ldap/ldap.conf :

♯ Le client utilise une liste d'AC de confiance
♯ pour valider l'identité du serveur. En effet, il vérifie
♯ que le certificat du serveur a bien été signé par
♯ la clé privée d'une de ces AC de confiance
♯ (ici, l'ac–serveur du CRU).
TLS_CACERT /etc/ssl/certs/ac–bundle.pem
♯ Le client doit obligatoirement valider l'identité
♯ du serveur.
TLS_REQCERT demand

De même que le serveur, le client utilisera un certificat délivré par l'ac-serveur du CRU. La clé publique du client lui permettra d'être authentifié par le serveur. Pour indiquer cela, on crée sur le client le fichier /home/user/.ldaprc, où *user* est le compte se connectant au serveur LDAP :

♯ Clé publique
TLS_CERT /etc/ssl/certs/client.crt
♯ La clé privée permet au client de déchiffrer
♯ les données que le serveur lui envoie via SSL/TLS.
♯ Elle est en lecture seule pour le compte user.
TLS_KEY /etc/ssl/private/client.key

3.5 Mise en application

le serveur maître est maitre.univ-paris1.fr et le réplicat replicat.univ-paris1.fr.

[1] Les exemples suivant ont été créés et utilisés sous *Debian GNU/Linux*.

Pour le maître :

Dans /etc/ldap/slapd.conf,

replogfile /var/lib/ldap/slapd.replog
replica host=replicat.univ–paris1.fr :389
suffix="dc=univ–paris1,dc=fr"
binddn="cn=replicat,dc=univ–paris1,dc=fr"
credentials="motdepasse"
bindmethod=simple
♯ Force l'utilisation de TLS entre slurpd
♯ sur le maître et slapd sur le réplicat.
tls=critical

Dans /etc/ldap/ldap.conf (utilisé par slurpd)

TLS_CACERT /etc/ssl/certs/ac–bundle.pem
♯ On demande également au client (slurpd)
♯ de toujours valider l'identité du serveur LDAP.
♯ (replicat.univ–paris1.fr).
TLS_REQCERT demand

Attention, slurpd tourne sous root, on crée donc le fichier /root/.ldaprc

TLS_CERT /etc/ssl/certs/maitre.univ–paris1.fr.crt
TLS_KEY /etc/ssl/private/maitre.univ–paris1.fr.key

Dans notre cas le couple certificat - clé privée du serveur maître sera aussi utilisé comme certificat - clé privée pour la connexion cliente de slurpd sur le réplicat, on a donc les permissions suivantes sur la clé privée du certificat :

```
-rw-r—— 1 root privkey 1636 (...) maitre.univ-paris1.fr.key
```

privkey étant un groupe système dont le compte slapd est membre. En effet, le démon slapd tourne sous le compte système slapd et peut lui aussi être un serveur SSL/TLS.

Remarque : Depuis l'arrivée des noyaux 2.6, le support des acls [7] est possible sur certains systèmes de fichiers comme EXT3. Ainsi les droits d'accès sur les certificats peuvent être définis très précisément. De plus, cette technique est recommandée par le CRU et sera mise en place prochainement à Paris 1.

Pour le réplicat :

Dans /usr/local/etc/openldap/slapd.conf :

♯Options TLS pour slapd
TLSCipherSuite HIGH :MEDIUM :+SSLv2
TLSCACertificateFile /etc/ssl/certs/ac–bundle.pem
TLSCertificateFile /etc/ssl/certs/replicat.univ–paris1.fr.crt
*TLSCertificateKeyFile *
/etc/ssl/private/replicat.univ–paris1.fr.key
♯ On oblige l'authentification du client slurpd
♯ tournant sur maitre.univ–paris1.fr.
TLSVerifyClient demand

Dans le cas d'une réplication via TLS le niveau de sécurité est donc correct.

Remarques :
- OpenLDAP ne supporte correctement **TLSVerifyClient demand** que depuis la version 2.2.23, 2.2.22 ne semblait pas supporter cette fonctionnalité ;
- Il faut aussi mettre en place des certificats du CRU pour CAS, pour Tomcat et pour l'ENT[2] ;
- Il reste encore à gérer les listes de révocation. Toutefois, OpenLDAP ne semble pas encore supporter cette fonctionnalité. De plus, contrôler les deux cotés de la communication (le client et le serveur) permet de réagir facilement à une corruption de certificat même sans liste de révocation.
- Il faut aussi interdire la réplication, où bien l'écriture sur les réplicats par une connexion n'utilisant pas TLS (cela doit être possible avec une ACL du type tls_ssf). En effet, si un *faux* client se connecte sans utiliser SSL/TLS, il peut écrire dans un réplicat simplement en connaissant le mot de passe de l'administreur LDAP.

4 Sécurité et optimisation de la base de données Berkeley DB

4.1 Préliminaires

Sous Debian, les paquetages libdb4.2 et db4.2-util permettent d'installer et de gérer la base de données BDB. db4.2-doc est lui aussi disponible.

4.2 Configuration

Le fichier DB_CONFIG, placé dans /var/lib/ldap/ avec les fichiers de la base de donnée, permet d'optimiser les performances de BDB.

Voici un exemple de configuration :

```
# Définit la taille de la mémoire cache allouée,
# L'efficacité de BDB croit avec la taille de la cache,
# jusqu'à un point d'efficacité maximum.
# Ici, 300 Mégabits de cache permettent à OpenLDAP
# d'avoir des temps de réponses rapides sans pénaliser
# la machine.
set_cachesize 0 300000000 0
#
# Si DB_TXN_NOSYNC est activé, Berkeley DB ne
# synchronisera pas immédiatement le disque avec
# des changements de la base en mémoire.
# Très utile pour maximiser les performances
# lors des écritures massives
# (préchargement de l'annuaire par exemple),
# il est préférable de le désactiver en usage normal
# (lecture majoritaire)
# afin de réduire le risque de perte de données
# en cas de "crash"
# set_flags DB_TXN_NOSYNC
#
# Il est important que les fichiers de logs soient
# sur un disque différent des fichiers de la base.
set_lg_dir /espace/bdb-logs
set_lg_regionmax 1048576
# Taille d'un fichier de journalisation
set_lg_max 10485760
set_lg_bsize 2097152
```

De plus, pour des bases de grandes tailles, il est impératif d'insérer régulièrement des points de contrôle[3]. En effet, cette opération offre un point de récupération de la base et permet de supprimer[4] les journaux des transactions antérieures (dans /espace/bdb-logs/log.*). Enfin, à Paris1, nous avons constaté que l'absence de points de contrôle pouvait corrompre la base de données, notamment lors de nombreux accès en écriture.

Dans /etc/slapd.conf :

```
database bdb
# Insère un checkpoint si 128 kilobits de données ont été
# écrits ou bien si 15 minutes se sont écoulées depuis
# le dernier point de contrôle
checkpoint 128 15
# dn de base
suffix "dc=univ-paris1,dc=fr"
```

Par ailleurs, il semblerait qu'un point de contrôle immédiat soit forcé par l'exécution de la commande slapcat.

4.3 Sauvegarde et maintenance de la base

La procédure de sauvegarde adoptée est très simple : toutes les nuits, le service slurpd est stoppé sur le maître, puis slapcat est lancé sur les réplicats et copie les données de l'annuaire dans un fichier ldif.

En cas de panne de l'annuaire dûe à une corruption des données de la base, exécuter *db4.2_recover -ve* après s'être placé dans /var/lib/ldap/ peut éviter un rechargement complet du ldif avec *slapadd*. Toutefois, on doit s'assurer que la base a été rétablie correctement en comparant les fichiers ldif extraits du serveur LDAP après réparation et d'un réplicat en fonctionnement.

Le chargement d'un ldif pour un réplicat en panne est aussi une opération délicate : il faudra gérer les modifications effectuées sur le maître lors du chargement. Cette manipulation est décrite en annexe.

[2] http://www.esup-portail.org/consortium/espace/SSO_1B/tech/cas/cas_x509.html
[3] man slapd-bdb
[4] par l'exécution de *db4.2_archive -d* dans /var/lib/ldap/

5 Un outil pour assurer la continuité du service CAS : *cas-spare*

5.1 Peut-on dormir sur ses deux oreilles avec un serveur CAS en production ?

CAS (Central Authentication Service) [8] est une solution de Single Sign On libre, simple, riche et sûre, développée par l'université de Yale [9], et adoptée par le projet ESUP-Portail [10]. Après avoir installé et configuré CAS avec les certificats du CRU, ce dernier devient alors l'unique point d'authentification du système d'information : **un arrêt de ce service de SSO interdit donc tout accès authentifié au système d'information**.

Or, CAS ne propose pas de système de répartition de charge ou de reprise sur pannes [11].

Toute la difficulté de mettre en place de tels systèmes vient du fait que les différents objets manipulés par CAS (essentiellement les tickets) sont maintenus en mémoire du processus serveur CAS, ceci à priori pour des raisons d'efficacité et de simplicité. Pour effectuer du partage de charge ou de la tolérance aux pannes, il est nécessaire de partager ces tickets [5] entre les différents serveurs.

La pratique montre que les universités ayant déployé CAS n'ont pas rencontré de problème de montée en charge, les processus mis en oeuvre étant relativement simples. Le problème de la tolérance aux fautes est en revanche, au vu de la criticité du service d'authentification, beaucoup plus crucial.

C'est dans cet objectif que l'utilitaire ***cas-spare*** [12], **distribué sous license LGPL, a été créé**. Il détecte un arrêt du service d'authentification et active alors une redondance de ce service dans un court délai. Il ne permet cependant pas de partager les tickets CAS entre différents serveurs.

5.2 D'autres solutions existent :

Heartbeat-drdb [13], permet à deux machines en cluster de se surveiller (Heartbeat) et de partager des données grâce à des disques en raid réseau (drdb). En cas de panne d'une des machines, c'est l'autre qui prend la relève. Cette solution est dite **générique** car elle peut s'appliquer à tous types de serveurs comme le DNS, la messagerie ou encore le web. Elle peut donc être utilisée par CAS.

Voici ses avantages :

- Cette solution est éprouvée et a fait ses preuves. Elle est supportée par une large communauté.
- En cas de panne, le service redémarre en moins de cinq secondes.
- Une fois que l'administrateur système l'a mise en place pour un service quelconque et a acquis une bonne connaissance de cette solution, il saura rapidement la mettre en place pour CAS. De plus, il sera déjà opérationnel pour effectuer les opérations de maintenance.
- Associée à un système de partage des tickets ainsi qu'à un bon outil de détection de pannes du service CAS, elle constituerait une solution excellente.

et ses inconvénients :

- La solution est difficile à prendre en main pour un administrateur système néophyte.
- Les utilisateurs doivent se réauthentifier en cas de panne du service CAS.
- Elle nécessite deux machines. De plus, la machine secondaire, qui peut néanmoins être peu puissante, n'est en fait quasiment pas utilisée.
- *Heartbeat* détecte uniquement si la machine primaire est allumée par ping sur câble croisé ou liaison série. Tous les types de pannes de CAS ne sont donc pas gérées. Par exemple, une panne de *Tomcat* sur le primaire n'est pas détectée, une panne de l'annuaire LDAP sous-jacent à CAS non plus.
- Autre désavantage, les machines ne peuvent pas être distantes.

Campus Crusade for Christ [14] distribue un serveur CAS modifié qui permet, couplé à la solution de partage de mémoire cache *JGroups*, de répliquer les tickets (cf. figure 4) entre les serveurs CAS. Cette solution permet ainsi d'effectuer de la répartition de charge entre plusieurs serveurs CAS. Elle n'est cependant pas une solution de reprise sur erreur : les clients peuvent en effet être dirigés vers un serveur en panne.

Figure 4 – *Un partage de tickets entre serveurs CAS permet d'obtenir une authentification persistante*

[5] En fait, il suffit de partager seulement les tickets rejouables : le *Ticket Granting Cookie* et le *Proxy Granting Ticket*. En effet, ce sont eux qui permettent d'obtenir les *Service Tickets* et les *Proxy Tickets*, non rejouables et limités dans le temps.

D'autres solutions sont à l'étude [15] :

Une base de données stockant les tickets (cf. figure 4) peut également remplacer *JGroups*. Les outils de bases de données, maîtrisés et connus, peuvent être maintenus facilement. Cette solution n'a jamais été implémentée mais CAS v3 propose des fonctionnalités permettant de stocker facilement les tickets dans une base de données.

Une autre idée est de créer un *Ticket Granting Cookie* qui ne soit pas une chaîne aléatoire mais qui contienne des informations chiffrées lui permettant d'être validé même en l'absence d'une entrée dans la cache. Cela signifie que, pour un ensemble de serveurs CAS E, un utilisateur authentifié sur un des serveurs de E pourra être reconnu par n'importe lequel des autres serveurs de E. Cette solution demanderait de fortes modifications du code de CAS. De plus, il faudrait tout de même interdire les authentifications multiples, en effet un ticket, même signé, ne doit pas pouvoir être réjoué indéfiniment.

Enfin, le nom du serveur CAS ayant réalisé l'authentification pourrait être inséré dans le ticket. Ainsi le client serait toujours renvoyé vers le bon serveur CAS. L'étude de cette solution, peu complexe à mettre en place, doit encore être approfondie. Toutefois, les tickets sont tout de même perdus en cas de panne d'un serveur CAS. Ainsi, utilisée seule, elle constitue uniquement une solution de répartition de charge.

5.3 Pourquoi *cas-spare* ?

La seule solution de reprise sur panne connue lors du démarrage du projet cas-spare est *Heartbeat-drdb*. Or nous ne maîtrisons pas ce produit et nous ne disposons pas d'une machine uniquement pour assurer la redondance de CAS. De plus, *Python* propose une fonction simple et très efficace pour surveiller l'activité de CAS, en simulant une connexion d'un utilisateur humain via un navigateur. L'administrateur système a aussi l'idée d'affecter plusieurs adresses IP à une même interface réseau, ce qui permet de joindre tous les serveurs CAS avec le même nom DNS : cas.yourdomainname.xy et de pouvoir ainsi intégrer un certificat du CRU pour ce service. C'est la naissance de *cas-spare*.

5.4 Une installation et configuration simple

cas-spare est simple à configurer comme vous pourrez le voir sur son site web [12].

Pour résumer, deux serveurs CAS, le principal et sa redondance, configurés avec le même certificat du CRU pour cas.yourdomainname.xy, sont placés dans le même VLAN. *cas-spare* est installé sur ces serveurs et est exécuté, par le compte root, en tâche de fond, toutes les minutes.

Attention, deux instances de *cas-spare* ne doivent pas s'exécuter en même temps sur deux serveurs différents. En effet, les deux machines pourraient tenter de devenir le serveur CAS principal simultanément. C'est pourquoi chaque serveur lancera *cas-spare* à 25 secondes d'intervalle.

Le service CAS sera accessible aux adresses IPv4 et IPv6 correspondant au nom DNS cas.yourdomainname.xy. Ces adresses vont rester fixes, ce qui évitera de mettre à jour cette entrée DNS, en effet une mise à jour du DNS est longue à répercuter sur l'ensemble de la toile.

cas-spare permet de mettre en place un nombre de redondances illimité. Cependant, le temps de reprise sur panne augmente avec le nombre de serveurs de spare. Si l'on considère qu'un serveur est indisponible 2 heures par an, alors le fait d'avoir 2 serveurs divise le nombre de chances que les deux serveurs soient indisponibles en même temps par 5000, et 3 serveurs par 25 millions. Disposer de 2 serveurs CAS est donc la solution à recommander.

5.5 Fonctionnement de *cas-spare*

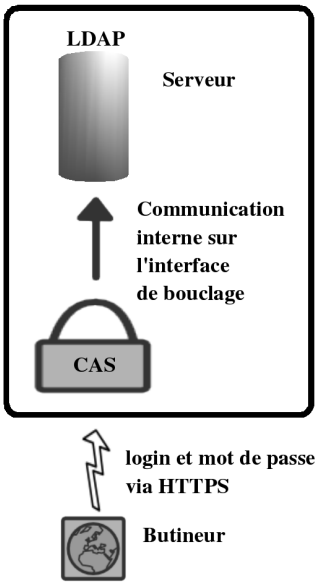

Figure 5 – *Authentification sécurisée sur un annuaire LDAP via CAS*

En production, nous disposons de deux serveurs CAS, un principal et une redondance. Les services CAS et LDAP sont installés en couple sur chaque machine, ainsi chaque service CAS utilise uniquement le service LDAP local comme on le voit dans la figure 5.

Remarque : CAS pourrait se connecter non seulement au serveur LDAP local mais aussi à des réplicats LDAP distants, via LDAPS. Cette technique permettrait de pallier à des pannes du service LDAP tout en conservant la confidentialité de l'identifiant et du mot de passe. Pour l'instant, nous n'avons pas mis en place ce mécanisme car il est beaucoup plus simple de configurer *CAS Generic Handler* [8] avec uniquement un annuaire LDAP local.

Par la suite, on désigne un serveur contenant CAS, LDAP, et *cas-spare* sous le terme de serveur d'authentification. Le

serveur utilisé à l'instant t pour l'authentification des utilisateurs du système d'information est nommé serveur principal. Chacun des serveurs possède sa propre adresse IPv4, le serveur principal utilise deux adresses IPv4 : la sienne, qui restera fixe, et celle correspondant au nom DNS cas.univ-paris1.fr, qui va changer de machine.

Voici un exemple (cf. figure 6) illustrant le fonctionnement de *cas-spare*. Soit 2 serveurs d'authentification A, et B avec A comme serveur CAS principal. *cas-spare* s'exécute sur A à l'instant t, sur B à $t + 25\ secondes$ puis de nouveau sur A à $t + 1\ minute$, et ainsi de suite.

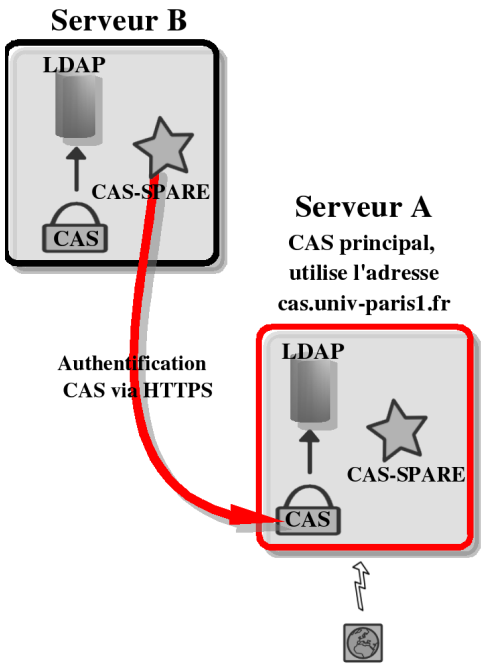

Figure 6 – *cas-spare interrogeant le serveur CAS principal*

Le diagramme de séquence UML représenté en figure 7 décrit un cas de reprise sur panne possible :

L'annuaire LDAP de A tombe en panne à $t + 30\ secondes$, entraînant une panne du service d'authentification. A $t + 25\ secondes$, *cas-spare* s'est exécuté sur B et a effectué une authentification réussie sur cas.univ-paris1.fr. A $t + 1\ minute$, A remarque que son serveur CAS local ne fonctionne pas et qu'il utilise l'adresse cas.univ-paris1.fr, il désactive donc cette adresse, redémarre CAS au cas où, et signale l'anomalie à l'administrateur système. A $t + 1\ minute + 25\ secondes$, B s'empare de l'adresse cas.univ-paris1.fr, met à jour la table ARP du routeur et rétabli le service CAS en IPv4 et en IPv6.

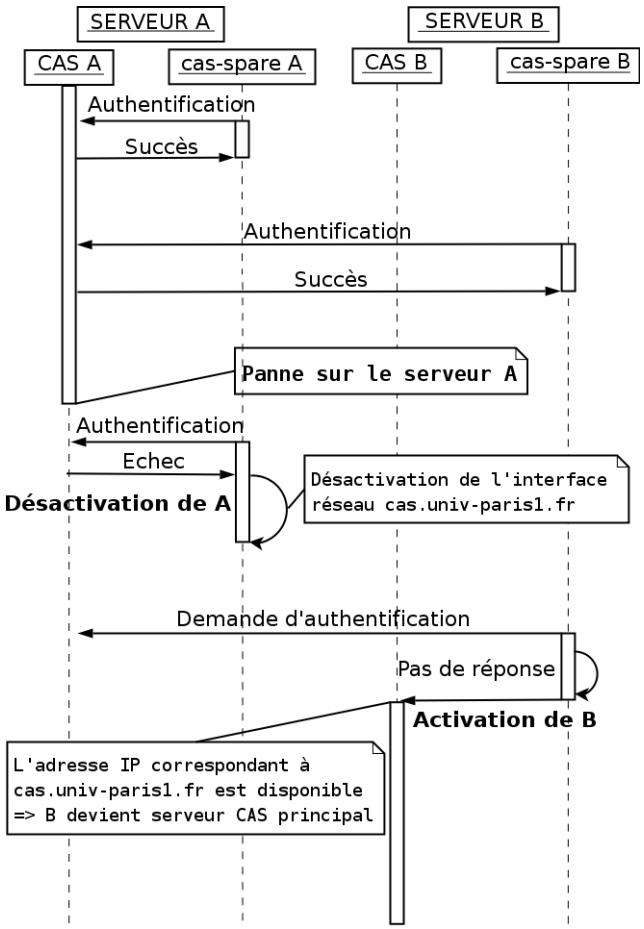

Figure 7 – *cas de reprise sur panne*

5.6 *cas-spare* : la meilleure solution de reprise sur pannes pour CAS ?

Voici les avantages de *cas-spare* :

- L'outil de surveillance du serveur CAS est très puissant. En effet, *cas-spare* simule une connexion utilisateur via un navigateur, il teste donc à la fois les services CAS et LDAP, mais aussi la connectivité au réseau, une panne ou un redémarrage de la machine ;
- *cas-spare* est simple à installer et à configurer.
- L'entrée DNS du serveur CAS ne change pas, cela évite les temps de propagation DNS en cas de remplacement du serveur CAS mais impose que les serveurs CAS soient dans le même VLAN. En outre, cette technique permet de n'avoir qu'un seul certificat du CRU se nommant par exemple cas.yourdomainname.xy.
- *cas-spare* est indépendant du service de SSO utilisé, il peut fonctionner avec d'autres SSO très facilement.
- *cas-spare* permet d'effectuer en souplesse la maintenance des serveurs. En effet, il assure la continuité du service CAS pendant la mise hors ligne d'un des serveurs. De plus ces machines peuvent exécuter simultanément d'autres applications et peuvent être composées d'architectures matérielles et logicielles complètement diffé-

rentes.
– Développé en language Python, il fonctionne sous des environnements Unix, Linux ou bien BSD.
– Il supporte des serveurs CAS tournant sous IPv4 seulement ou en double pile IPv4/IPv6.
– Il est disponible sous une license LGPL [12], et son code d'environ 500 lignes est commenté et facile à comprendre.

et ses limitations :

– Les utilisateurs devront se réauthentifier après une panne du serveur CAS d'origine. En effet leur ticket ne sera pas accepté par la redondance.
– Le délai de reprise sur panne est de moins d'une minute trente. Cependant, il peut encore être facilement divisé par deux ou trois après optimisation de l'application.
– *cas-spare* doit être installé sur chaque serveur CAS. Il s'installe en revanche très facilement.
– C'est un produit jeune et des erreurs de programmation peuvent encore être découvertes.

De plus, *cas-spare* ne fonctionne que si toutes les redondances à CAS sont dans le même VLAN, ce qui ne permet pas de gérer une panne du coeur de réseau ou du routeur. Cela n'est pas primordial car le plus souvent la plupart des serveurs d'applications sont dans le même VLAN (DMZ) que CAS, une panne du coeur de réseau affecte alors l'ensemble des services.

La méthode de détection des pannes de *cas-spare* peut détecter une panne d'un serveur CAS situé dans un autre VLAN, *cas-spare* pourrait donc gérer des redondances distantes comme c'est le cas pour, par exemple, un service DNS. Toutefois, il faudrait pour cela que les clients CAS puissent être configurés avec une liste de serveurs CAS (par exemple, cas1.yourdomainname.xy, cas2.youdomainname.xy et cas3.yourotherdomain.xy), comme on peut le faire avec un client LDAP ou bien un client DNS. Or, modifier l'ensemble des clients CAS pour leur permettre d'accéder à une liste de serveurs CAS demande de forts développements et pose d'autres problèmes : le client doit notamment détecter les serveurs CAS en panne avant la redirection. Or cette manoeuvre implique des temps de réponses importants et pourrait nuire à l'efficacité du service d'authentification. Cependant, nous pensons que cette solution, adoptée par LDAP et DNS, mérite d'être étudiée plus avant.

Le tableau suivant présente un bref comparatif des solutions :

	Heartbeat-drdb	**JGroups**	**cas-spare**
Répartition de charge	Possible	Oui	Possible
Reprise sur panne	Oui	Non	Oui
Réauthentification	Oui	Non	Oui
Qualité du monitoring	Faible	X	Excellente
Temps de reprise	$<$ 5 sec	X	$<$ 1 min 30
Machines distantes	Non	?	Oui
VLANs $\neq$	Non	?	Possible
Prise en main	Difficile	?	Simple
Coût matériel	Elevé	Très faible	Très faible

La solution de **Campus Crusade for Christ**, basée sur *JGroups* n'est pas une solution de reprise sur panne et est encore peu diffusée, c'est pourquoi elle ne répond pas à certaines fonctionnalités ou est inconnue sur certains aspects.

5.7 *cas-spare* fonctionne !

cas-spare est en service à l'université Panthéon-Sorbonne depuis début juin 2005 et pour l'instant aucune panne du système d'authentification n'est apparue. Une analyse des fichiers journaux produits par *cas-spare* durant les mois de juillet, août et septembre, montre que la seule panne du service CAS a été reprise dans un délai de 20 secondes, en septembre. La panne n'était pas dûe à CAS lui-même mais à un arrêt de l'annuaire LDAP en vue d'indexer sa base de données.

Afin d'avoir un aperçu de la réelle efficacité de *cas-spare*, plusieurs types de pannes ont été simulées sur nos serveurs en production.

Heure	**Serveur**	**Evènement**
15 :21 :31	A	Arrêt de CAS
15 :22 :23	B	Activation du CAS principal
15 :23 :35	B	Arrêt de slapd
15 :24 :55	A	Activation du CAS principal
15 :25 :58	A	Arrêt de la machine
15 :26 :21	B	Activation du CAS principal
15 :31 :41	B	Arrêt violent de Tomcat
15 :32 :56	A	Activation du CAS principal
15 :34 :23	A	Coupure du lien réseau
15 :35 :22	B	Activation du CAS principal

cas-spare reprend donc l'ensemble des pannes simulées, dans un délai raisonnable. La version suivante de *cas-spare* devrait diviser par deux, voire même trois, les temps de reprise sur erreur. En effet, la borne supérieure du temps d'exécution de *cas-spare*, dûe à des délais d'attente en cas de non réponse à une requête réseau (*time-out*), a été surévaluée dans la première version du logiciel. Une optimisation de l'algorithme de *cas-spare* conduira donc a une conséquente réduction du délai de reprise sur erreur.

5.8 Limitations actuelles et perpectives

La principale limitation de **cas-spare** est de forcer la réauthentification des utilisateurs après chaque panne du service CAS. Il n'effectue en effet qu'une bascule entre deux serveurs CAS. Pour permettre une authentification persistante, *cas-spare* doit être couplé à une technique de partage de tickets.

La solution consistant à signer les *Tickets Granting Cookie* pour qu'il puissent être reconnus par l'ensemble des serveurs CAS redondants et protégés par *cas-spare* serait acceptable. Elle n'est cependant pas implémentée, car trop coûteuse en développement. En outre, elle doit être associée à une technique permettant de limiter la durée de vie des tickets signés : en effet, rejouer un ticket signé ne doit pas permettre de s'authentifier indéfiniment.

Une alternative serait d'indiquer dans chaque ticket par quel serveur CAS il a été délivré, et de rediriger le client vers ce même serveur. Par contre, les tickets d'un serveur CAS en panne seraient tout de même perdus, obligeant les utilisateurs à se réauthentifier. Cette solution ne permet donc que du partage de charge.

Une autre solution est de partager les tickets CAS rejouables (TGC et PGT) via une base de données ou un partage réseau. On repousse cependant les problèmes de fiabilité sur ces derniers, en effet une panne de la base de données ou du partage réseau bloque l'ensemble du service CAS.

Toutefois, les outils de base de données sont bien connus et bénéficient eux-aussi d'outils de tolérance aux fautes. Une solution constituée de CAS, de *cas-spare*, et d'une base de données stockant les tickets serait donc efficace. De plus, elle permettrait de faire à la fois de la reprise sur erreur et de la répartition de charge. Enfin, l'architecture de CAS v3 facilite la mise en place de cette technique.

6 Conclusion

La réplication des annuaires, point sensible, s'effectue de manière sûre en empêchant à la fois la corruption et la recopie frauduleuse des données. De plus, l'annuaire LDAP est facilement rétabli en cas de panne grâce à une technique de sauvegarde éprouvée.

Le service CAS offre une forte disponibilité : 7500 étudiants se sont inscrits par internet en septembre 2005, et plus de 20000 inscriptions sont attendues d'ici décembre. Toutes les pannes survenues sont reprises par *cas-spare* sans aucune intervention humaine. *cas-spare* constitue donc un outil simple et efficace de reprise sur pannes et je vous invite à l'utiliser.

Les techniques proposées dans cet article sécurisent votre service d'authentification et sont rapides et faciles à appliquer. Une fois mises en place, elles vous permettront d'installer en toute sérénité d'autres services dans votre environnement numérique de travail.

Références

[1] Schéma directeur des espaces numériques de travail. http://www.educnet.education.fr/equip/sdet.htm.

[2] Gerald Carter. LDAP, System Administration. O'REILLY, 2003.

[3] Jean-Luc Archimbaud. Certificats (électroniques), Pourquoi, Commment ? Rapport technique, CNRS/UREC, 2000. http://www.urec.cnrs.fr/securite/articles/certificats.kezako.pdf.

[4] Autorité de certification du CRU. http://igc.cru.fr/.

[5] Serge Aumont, Claude Gross, Philippe Leca. Certificats X509 et Infrastucture de Gestion de Clés. Dans *Actes de la conférence JRES2001*, Lyon, 2001. http://www.cru.fr/igc/JRES01.tutoriel.IGC.pdf.

[6] Fabrice Jammes. Configuration d'OpenLDAP pour réplication via TLS. http://www.cru.fr/faqdata/cache/112.html, 2005. note technique dans la FAQ du CRU.

[7] Linux Extended Attributes and ACLs. http://acl.bestbits.at/.

[8] CAS, Generic Handler. http://esup-casgeneric.sourceforge.net/.

[9] Drew Mazurek. uPortal and the Yale Central Authentication Service. Dans *Acte de la conférence JA-SIG Summer Conference '04*, Denver, CO, Juin 2004.

[10] *ESUP-Portail*, espace numérique de travail d'accès intégré aux services pour les étudiants et le personnel de l'enseignement supérieur, section destinée à cas. http://www.esup-portail.org/consortium/espace/SSO_1B/cas/index.html.

[11] Pascal Aubry, Vincent Mathieu, Julien Marchal. Single Sign-On open-source avec CAS (Central Authentication Service). Dans *Actes de la conférence JRES2003*, Lille, Décembre 2003. http://2003.jres.org/actes/paper.139.pdf.

[12] *cas-spare*, un outil de reprise sur erreur pour un service de SSO. https://sourcesup.cru.fr/projects/cas-spare/.

[13] Mise en place de *Heartbeat-drdb* au CRIUM de Metz. http://www.crium.univ-metz.fr/docs/system/drbd/.

[14] Campus Crusade for Christ, CAS server distribution. http://gcx1.mygcx.org/cas/CCCIChanges.html.

[15] JASIG (Java Architectures Special Interest Group), documentation and collaboration site. https://clearinghouse.ja-sig.org/wiki/.

Annexes

Rechargement d'un réplicat à la suite d'une panne

Nous décrivons ici une méthode permettant de remettre en route par rechargement intégral un serveur LDAP réplicat dont la base BDB est endommagée, sans compromettre la synchronisation maître/réplicat ni interrompre le service.

Notre configuration est constituée d'un serveur maître (master) et de deux réplicats (replica1 et replica2) alimentés par slurpd. Le serveur endommagé est replica1.

Points à noter :
- La configuration est réalisée sur des plates-formes Debian Gnu/Linux Sarge.
- Sur le maître, le script de lancement /etc/init.d/slapd a été modifié pour accepter les paramètres "start_slurpd" et "stop_slurpd" permettant de contrôler le démon slurpd indépendamment de slapd.
- Sur les réplicats, le script "backup.sh" lance une sauvegarde par "slapcat" de la base LDAP vers un fichier horodaté placé dans /espace/ldap.
- Le démon slapd fonctionne sous le compte UNIX "slapd".

1) arrêter slapd sur replica1, en lui forçant la main si nécessaire, et vérifier qu'aucun processus slapd ne reste actif :
replica1♯ /etc/init.d/slapd stop
replica1♯ ps -ef | grep slapd
replica1♯ kill -9 .....

2) arrêter slurpd sur le maître :
master♯ /etc/init.d/slapd stop_slurpd

3) créer une sauvegarde ldif du réplicat valide avec slapcat. L'opération ne prend pas plus de 3 minutes :
replica2♯ /home/slapd/ldap-tools/backup.sh

4) copier ce ldif sur le réplicat endommagé :
replica2♯ cd /espace/ldap
replica2♯ sftp moncompte@replica1
—> put ldap-date-heure.ldif

5) sur le maître, aligner l'horodatage slurpd du réplicat endommagé sur celui du réplicat valide (en effet une fois le ldif chargé, le réplicat endommagé se retrouvera en l'état actuel du réplicat valide) :
master♯ cd /var/spool/slurpd/replica
master♯ vi slurpd.status
«< exemple d'ancien slurpd.status :
replica1.univ-paris1.fr :389 :1114105266 :10
replica2.univ-paris1.fr :389 :1114161845 :1
»> modifier ce fichier donnera :
replica1.univ-paris1.fr :389 :1114161845 :1
replica2.univ-paris1.fr :389 :1114161845 :1

6) redémarrer slurpd :
master♯ /etc/init.d/slapd start_slurpd

7) sur le réplicat endommagé, supprimer tous les fichiers et logs de la base BDB :
replica1♯ cd /usr/local/openldap-data
replica1♯ rm *.bdb __*
replica1♯ cd /espace/bdb-logs
replica1♯ rm *.log

8) puis lancer la restauration de la base et démarrer slapd une fois terminé. La restauration prend environ 1h.
replica1♯ su slapd -c '/usr/sbin/slapadd -b "dc=univ-paris1,dc=fr" -l /home/moncompte/ldap-date-heure.ldif' ; sleep 240 ; /etc/init.d/slapd start

Remarque : pour minimiser le temps d'exécution de slapadd, il est possible d'activer le drapeau DB_TXN_NOSYNC dans le fichier DB_CONFIG. Il devra être de nouveau désactivé après la fin d'exécution du slapadd et avant le lancement de slapd.