

OpenLDAP au quotidien: trucs et astuces

Guillaume Rousse
Ingénieur système à l'INRIA

novembre 2009

OpenLDAP¹ est l'implémentation d'annuaire LDAP la plus utilisée dans le monde du libre à l'heure actuelle. Malheureusement, la documentation disponible n'est pas vraiment à la hauteur. En particulier, les fonctionnalités offertes par les différents modules, qu'il s'agisse de greffons (overlays), ou de modules de stockage (backend), se réduisent le plus souvent à une page de manuel compréhensible par les seuls initiés, qui sont de plus les seuls à en connaître l'existence. Il s'ensuit un gouffre relativement important entre les bonnes pratiques qui se transmettent de bouche à oreille sur la liste de diffusion des utilisateurs, et l'utilisation classique que l'on voit partout ailleurs. Plusieurs années de pratique m'ont ainsi permis d'accumuler une certaine expérience sur ce logiciel, pour répondre à des problèmes variés liés à la gestion d'une base d'utilisateur dans un annuaire LDAP. Ces problèmes étant susceptibles de se poser pour tout administrateur système se trouvant dans le même cas, j'ai donc choisi de partager cette expérience sous la forme d'une liste de recettes thématiques.

Deux remarques au préalable. D'abord, ces astuces ne concernent pas OpenLDAP stricto-sensu, mais plutôt l'utilisation d'OpenLDAP pour gérer des comptes Unix, donc également `pam_ldap` et `nss_ldap`. Ensuite, OpenLDAP étant largement modulaire, la façon dont il est compilé influence largement la disponibilité de certaines fonctionnalités, qui peuvent être absentes, présentes dans l'exécutable principal, ou sous forme de modules externes, qui doivent être chargés explicitement par la directive `moduleload`. Attention, l'option de compilation `--enable-overlays` ne concerne que les modules principaux, d'autres sont également disponibles dans le répertoire `contrib/slapd-modules`, et doivent être compilés manuellement.

1 Sécurité

La sécurité est le premier thème abordé ici. Les recettes présentées concernent différentes stratégies visant à limiter l'exposition des mots de passes, la limitation des privilèges, la protection contre les dénis de service, la mise en place d'autres modes d'authentification.

1. <http://www.openldap.org>

1.1 Retrait des mots de passe dans les fichiers

Le stockage de mot de passes dans des fichiers, que ce soit en clair ou chiffrés, est une faille de sécurité potentielle, même si ce fichier est protégé par des droits d'accès. Dans la mesure du possible, il vaut mieux l'éviter. Or dans une utilisation classique d'OpenLDAP, il y a deux endroits où l'on trouve de tels mots de passes :

- dans la configuration de slapd
- dans la configuration de pam_ldap

Le mot de passe dans la configuration de slapd (`/etc/openldap/slapd.conf` ou `/etc/ldap/slapd.conf`, en fonction des distributions) correspond à un super-utilisateur, spécifié par les directives `rootdn` et `rootpw`. Comme l'utilisateur root d'un système Unix, cet utilisateur ignore les différentes restrictions, comme les ACLs par exemple, ce qui le rend pratique en cas de verrouillage grave de l'annuaire, par exemple. Cependant, à quelques exceptions près (serveur esclave, par exemple), son utilisation n'est absolument pas obligatoire, et peut facilement être remplacée par un compte normal stocké dans l'annuaire, doté des ACLs nécessaire pour lui donner les même privilèges.

La configuration de slapd ressemble à ceci :

```
# décommenter en cas d'urgence
#rootdn "cn=root,ou=roles,dc=domain,dc=tld"
#rootpw s3cr3t

# le mot de passe est un attribut sensible
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" write
    by self write
    by anonymous auth
    by * none

# les autres attributs le sont moins
access to dn.subtree="dc=domain,dc=tld"
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" write
    by self write
    by * read
```

Le mot de passe dans la configuration de pam_ldap (fichier `/etc/ldap.secret`) correspond à un utilisateur spécifié par la directive `rootbinddn` dans le fichier de configuration principal (`/etc/ldap.conf`), permettant au compte root du système de s'authentifier auprès de l'annuaire quand il a besoin d'y effectuer des opérations privilégiées. La seule opération de ce type permise par PAM étant le changement des mot de passe, le fait de ne pas utiliser cette fonctionnalité n'a comme seule conséquence que d'empêcher de changer le mot de passe d'un utilisateur sans en connaître la valeur précédente en utilisant la commande `passwd` depuis le compte root, comme s'il était stocké localement.

1.2 Interdiction des mots de passe en clair sur le réseau

Éliminer les mots de passes des fichiers n'est qu'une première étape, encore faut-il s'assurer qu'ils ne circulent pas librement sur le réseau, où il est si facile de les récupérer au passage.

Une solution simple consiste à interdire toute opération (écriture, lecture ou authentification) faisant intervenir ces mots de passes sur une connexion non chiffrée, par le biais d'une ACL dédiée. La valeur numérique utilisée pour le paramètre `ssf` (Security Strength Factor) est la force minimale du chiffrement nécessaire, à ajuster en fonction de vos applications (56 correspond à DES, 112 au triple DES, 128 au chiffrements forts actuels, voire `slapd.conf(5)` pour les détails). Voici la configuration nécessaire :

```
# interdiction d'accès au mot de passe sur une connexion non chiffrée
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" ssf=56 write
    by self ssf=56 write
    by anonymous ssf=56 auth
    by * none
```

Une solution moins radicale consiste à faire des exceptions pour certaines situations jugées suffisamment sûres, comme notamment l'hôte local, pour éviter par exemple de se faire systématiquement refuser la connexion en raison d'un certificat dont l'adresse ne correspond pas... :

```
# interdiction d'accès au mot de passe sur une connexion non chiffrée
# sauf depuis l'hôte local
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" ssf=56 write
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" peername.ip=127.0.0.1 write
    by self ssf=56 write
    by self peername.ip=127.0.0.1 write
    by anonymous ssf=56 auth
    by anonymous peername.ip=127.0.0.1 auth
    by * none
```

Attention, cette solution ne fait qu'interdire l'accès au mot de passe dans un contexte non sécurisé, mais n'empêche pas un client mal configuré d'envoyer le mot de passe en question...

1.3 Disparition des comptes partagés

L'utilisation d'un compte privilégié commun pour l'administration de l'annuaire, dans le cas d'un travail en équipe, nécessite généralement le partage du mot de passe de ce compte. Ce qui revient souvent à voir ce mot de passe conservé dans d'autres fichiers, échangé avec les nouveaux arrivants, etc... Éliminer les mots de passe de la configuration LDAP pour les garder stockés ailleurs ne

fait donc que déplacer le problème de dissémination de l'information sensible, pas le résoudre.

Une solution consiste alors à attribuer les privilèges d'administration directement aux comptes utilisateurs de l'ensemble de l'équipe d'administration, qui par définition ne sont pas partagés, et à supprimer le compte partagé devenu inutile. Traduite en terme d'ACLs, cette solution se matérialise ainsi :

```
access to dn.subtree="dc=domain,dc=tld"
    by dn="uid=foo,ou=users,dc=domain,dc=tld" write
    by dn="uid=bar,ou=users,dc=domain,dc=tld" write
    by self write
    by * read
```

Bien évidemment, l'énumération des comptes concernés devient très vite fastidieuse est difficile à maintenir. L'utilisation d'une ACL de groupe, comme détaillé à la recette 2.1, est largement préférable.

Les implications de cette solution sont assez importantes. En terme de sécurité, d'abord, il n'y a plus un seul compte privilégié, mais plusieurs, ce qui augmente la surface d'attaque potentielle. Ensuite, elle augmente les risques en cas de manipulation erronées de ces utilisateurs, puisqu'il n'y a plus de restrictions. Ce qui fait dire à certains qu'elle revient à la pratique largement déconseillée de travailler sous l'identité root en permanence. Cette comparaison n'est cependant pas exacte. En effet, cette augmentation de privilège n'est vrai qu'en mode authentifiée, alors que le mode anonyme est généralement suffisant pour les opérations de lecture. De plus, ces opérations se font sous une identité propre, ce qui offre de meilleures garanties de traçabilité (notamment si une journalisation des modifications est en place, comme expliqué à la recette 7.1). Il est donc plus réaliste de comparer avec l'utilisation de sudo.

1.4 Limitation des privilèges

La stratégie présentée jusqu'ici consiste à éviter d'exposer les mots de passes de comptes privilégiés. Néanmoins, cette politique s'accompagne parfois de pertes de fonctionnalités (voire l'exemple du changement de mot de passe par PAM), et se révèle même d'autre fois impossible à tenir. Bref, comment faire quand l'utilisation d'un mot de passe est inévitable ?

Une solution alternative consiste à utiliser des comptes dédiés, par opposition à un compte d'administration générique, dotés du minimum de privilèges nécessaires, de façon à diminuer les conséquences en cas de compromission. Cette solution a de plus l'avantage d'améliorer la traçabilité des changements, lorsqu'elle est employée en conjonction avec la recette 7.1.

Dans le cas précédent de pam_ldap, le seul privilège nécessaire consiste à pouvoir modifier le mot de passe d'un utilisateur. La configuration de slapd est à modifier de la façon suivante :

```
# le mot de passe n'est modifiable que par l'admin ldap, root
# à travers pam et son propriétaire
```

```

access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" write
    by dn="cn=pam,ou=roles,dc=domain,dc=tld" write
    by self write
    by anonymous auth
    by * none

```

1.5 Protection contre les dénis de service

Les ACLs permettent de spécifier qui peut avoir accès à quoi, mais pas en quelle quantité. Le mode d'interrogation sans authentification (anonyme), en particulier, est une porte ouverte potentielle aux attaques par déni de service, à moins bien sûr d'avoir une confiance absolue dans les seules machines capables d'ouvrir une connexion vers le serveur. Comment s'en protéger ?

La mise en place de limites explicites à l'utilisation des ressources permet de combler cette faiblesse. OpenLDAP fournit deux types de limites : les limites quantitatives gouvernent la quantité maximale d'entrées retournées par une requête, tandis que les limites temporelles gouvernent le temps maximal (en secondes) que slapd consacre à une requête. Il y a également deux types de critères d'application : les limites souples s'appliquent aux clients qui ne précisent aucune limite dans leur requête, tandis que les limites dures s'appliquent aux clients qui spécifient une limite explicite. Dans le cas des limites quantitatives, il est également possible de limiter le nombre d'enregistrements examinables avant filtrage de façon interne, pour éviter par exemple une attaque sur un attribut non indexé. Comme pour les ACLs, il est possible de préciser de manière assez fine à qui s'appliquent ces limites, ce qui permet de créer des groupes de privilèges échelonnés, comme dans l'exemple de configuration ci-dessous :

```

# utilisateur utilisé pour la synchronisation
limits dn="cn=sync,ou=roles,dc=domain,dc=tld"
    size.soft=unlimited size.hard=unlimited size.unchecked=unlimited
    time.soft=unlimited time.hard=unlimited

# administrateurs
limits group="cn=admins,ou=groups,dc=domain,dc=tld"
    size.soft=unlimited size.hard=unlimited size.unchecked=unlimited
    time.soft=unlimited time.hard=unlimited

# autres utilisateurs authentifiés
limits users
    size.soft=1024 size.hard=2048 size.unchecked=32767
    time.soft=15 time.hard=30

# requêtes anonymes
limits anonymous

```

```
size.soft=512 size.hard=1024 size.unchecked=32767
time.soft=10 time.hard=20
```

Attention, il est important de s'assurer que le compte utilisé pour la synchronisation n'est pas limité, sous peine d'avoir des serveurs secondaires incomplets. De même, il faut faire attention aux éventuelles implications sur `nss_ldap`, qui utilise par défaut une connexion anonyme : si celles-ci sont limitées, et que le nombre d'utilisateurs ou de groupes dépasse cette limite, les appels systèmes `getpwent`, `getgrent` et consort auront des résultats tronqués.

Pour plus d'information, consulter :

- la page de manuel `slapd.conf`

2 Règles de contrôle d'accès

Les règles de contrôle d'accès, ou ACLs (Access Control Lists) pour les initiés, réglementent qui a accès à quoi, de quelle manière, et sous quelle condition. Les recettes qui suivent montrent d'abord des astuces syntaxiques facilitant la maintenance de règles classiques, puis comment mettre en place des règles plus exotiques.

Pour plus d'information concernant les recettes de cette section, consulter :

- la page de manuel `slapd.access(5)`
- la page contrôle d'accès du guide d'administration d'OpenLDAP²

2.1 Règles de groupe

Une règle applicable à plusieurs identifiants, telles que proposée dans la recette 1.3, devient très vite fastidieuse à maintenir s'il faut énumérer ces identifiants. Surtout si plusieurs règles sont concernées, et qu'il faut s'assurer de l'identité des listes à chaque fois.

Comme en administration système, l'utilisation d'un groupe LDAP pour formuler cette règles est bien plus simple :

```
access to dn.subtree="dc=domain,dc=tld"
  by group="cn=ldap_admins,ou=groups,dc=domain,dc=tld" write
  by self write
  by * read
```

Un groupe LDAP est une entrée appartenant à la classe `groupOfNames`, et ses membres sont identifiés par leur DN. Voici l'exemple d'une telle entrée, au format LDIF :

```
dn: cn=ldap_admins,ou=groups,dc=domain,dc=tld
objectClass: groupOfNames
cn: ldap_admins
member: uid=foo,ou=users,dc=domain,dc=tld
member: uid=bar,ou=users,dc=domain,dc=tld
```

2. <http://www.openldap.org/doc/admin24>

De façon plus générale, n'importe quelle classe peut être utilisée, pourvu que ses membres soit référencés par leur DN, au prix de quelques détails syntaxiques. En particulier, les groupes dynamiques présentés à la recette ?? peuvent également être utilisés. Quand au problème de recouvrement entre ces groupes LDAP, et les groupes POSIX, définies elles par la classe **posixGroup**, la recette 4.2 permet d'y remédier.

2.2 Règles d'attributs

Une règle destinée à s'appliquer à certains attributs d'une entrée nécessite l'énumération de ceux-ci. Lorsque ces attributs sont nombreux, ceci devient vite fastidieux. Par exemple, Horde définit pour chacune de ses applications susceptibles de stocker des informations personnelles un attribut : **impPrefs**, **turbaPrefs**, **hordePrefs**, etc...

Une première solution, si les noms de ces attributs possèdent une partie commune, consiste à utiliser les expressions régulières pour désigner l'ensemble de ces attributs :

```
access to dn.subtree="ou=users,dc=domain,dc=tld"
    attrs.regex="[a-z]+Prefs"
    by self write
    by * read
```

Une autre solution est de désigner ces attributs par la classe à laquelle ils appartiennent, ce qui garantit de n'en oublier aucun :

```
access to dn.subtree="ou=users,dc=domain,dc=tld"
    attrs=@hordePerson
    by self write
    by * read
```

2.3 Règles nominatives

La déclaration des ACLs dans OpenLDAP se fait par cible, c'est-à-dire que l'on déclare d'abord une partie de l'annuaire, éventuellement une liste d'attributs, puis qui peut intervenir dessus, puis comment (lire, écrire, etc). Quand le nombre d'intervenants potentiel augmente, par exemple suite à la mise en place d'une stratégie de privilèges minimums comme présenté dans la recette 1.4, ceci devient rapidement difficile à gérer. Dans l'exemple donné plus haut pour l'attribut mot de passe, si la réplication utiliser un identifiant dédié **cn=syncrepl,ou=roles,dc=domain,dc=tld**, il est facile de constater que l'annuaire secondaire sera dépourvu des mots de passe, puisqu'il n'aura pas accès à ceux-ci en lecture sur l'annuaire maître :

```
# interdiction d'accès au mot de passe sur une connexion non chiffrée
# sauf depuis l'hôte local
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
```

```

by dn="cn=admin,ou=roles,dc=domain,dc=tld" ssf=56 write
by dn="cn=admin,ou=roles,dc=domain,dc=tld" peername.ip=127.0.0.1 write
by self ssf=56 write
by self peername.ip=127.0.0.1 write
by anonymous ssf=56 auth
by anonymous peername.ip=127.0.0.1 auth
by * none

```

Une solution consiste dans ce cas à regrouper les déclarations d'ACLs par intervenant, en tirant avantage du paramètre **break**, qui permet de poursuivre la lecture de ces règles pour un intervenant non cité, plutôt que de supposer que s'il n'est pas cité, il n'a aucun accès. L'exemple précédent devient alors :

```

# admin: accès en écriture global
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" ssf=56 write
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" peername.ip=127.0.0.1 write
    by * break

access to dn.subtree="dc=domain,dc=tld"
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" write
    by * break

# syncrepl: accès en lecture global
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by dn="cn=syncrepl,ou=roles,dc=domain,dc=tld" ssf=56 read
    by dn="cn=syncrepl,ou=roles,dc=domain,dc=tld" peername.ip=127.0.0.1 read
    by * break

access to dn.subtree="dc=domain,dc=tld"
    by dn="cn=admin,ou=roles,dc=domain,dc=tld" read
    by * break

# autres intervenants
access to dn.subtree="dc=domain,dc=tld" attrs=userPassword
    by self ssf=56 write
    by self peername.ip=127.0.0.1 write
    by anonymous ssf=56 auth
    by anonymous peername.ip=127.0.0.1 auth
    by * none

access to dn.subtree="dc=domain,dc=tld"
    by * read

```

Le résultat n'est certainement pas plus compact, mais généralement plus facile à lire et à comprendre, ce qui sur un sujet sensible comme les règles de contrôle d'accès, n'est pas un mince avantage.

2.4 Règles relationelles

Il est facile de mettre en place des ACLs basées sur un identifiant statique, autorisant par exemple l'administrateur (ou un groupe d'administrateurs, comme présenté dans la recette précédente) à modifier tout ou partie de l'annuaire. Il est également possible d'utiliser un identifiant dynamique comme **self** dans une ACL pour désigner l'utilisateur courant, typiquement pour autoriser une personne à modifier sa propre entrée. Comment autoriser un tiers, variable en fonction de l'entrée, à modifier celle-ci, notamment pour les entrées ne correspondant pas à des personnes ?

OpenLDAP permet de le faire simplement en utilisant la valeur d'un attribut de l'entrée, du type DN, pour désigner le tiers en question, par le paramètre **dnattr**. Il reste donc à trouver quel attribut utiliser en fonction du cas.

Certaines classes contiennent déjà de tels attributs dédiés, il suffit de les utiliser. L'exemple suivant autorise ainsi le responsable hiérarchique d'une personne, désigné par l'attribut **manager** de la classe **person** à modifier certains attributs administratifs de celle-ci :

```
# attributs modifiables par les administrateurs, le responsable
# ou la personne concernée
access to dn.subtree="ou=users,dc=domain,dc=tld"
    attrs=telephoneNumber,mobile,facsimileTelephoneNumber,roomNumber,postalAddress
    by group="cn=admins,ou=roles,dc=domain,dc=tld" write
    by dnattr=manager write
    by self write
    by * read
```

Dans le cas ci-dessous, l'utilisateur q peut ainsi modifier les attributs de l'utilisateur james :

```
# l'utilisateur
dn: uid=james,ou=users,dc=domain,dc=tld
objectClass: inetOrgPerson
uid: james
manager: uid=q,ou=users,dc=domain,dc=tld

# son responsable
dn: uid=q,ou=users,dc=domain,dc=tld
objectClass: inetOrgPerson
uid: q
```

Pour les autres, il suffit d'utiliser l'attribut opérationnel **owner**, présent automatiquement pour n'importe quelle entrée. Cet autre exemple autorise le responsable d'un groupe à modifier la liste des membres de celui-ci :

```
# attributs modifiables par les administrateurs ou le responsable du groupe
access to dn.subtree="ou=groups,dc=domain,dc=tld" attrs=member
    by group="cn=admins,ou=roles,dc=domain,dc=tld" write
```

```
by dnattr=owner write
by * read
```

Dans le cas ci-dessous, l'utilisateur q peut ainsi gérer la liste des membres du groupe 00 :

```
# le groupe
dn: cn=00,ou=groups,dc=domain,dc=tld
objectClass: groupOfNames
cn: 00
member: uid=james,ou=users,dc=domain,dc=tld
member: uid=joe,ou=users,dc=domain,dc=tld
owner: uid=q,ou=users,dc=domain,dc=tld

# son responsable
dn: uid=q,ou=users,dc=domain,dc=tld
objectClass: inetOrgPerson
uid: q
```

Dans les deux cas, on reste cependant au cas d'une référence directe d'une entrée vers une autre. Si l'on veut passer à un modèle indirect, il faut utiliser les ensembles, avec le paramètre **set**. L'exemple suivant ajoute la secrétaire du responsable d'une personne à l'ACL vue plus haut :

```
# attributs modifiables par les administrateurs, le responsable,
# sa secrétaire, ou la personne concernée
access to dn.subtree="ou=users,dc=domain,dc=tld"
    attrs=telephoneNumber,mobile,facsimileTelephoneNumber,roomNumber,postalAddress
    by group="cn=admin,ou=roles,dc=domain,dc=tld" write
    by dnattr=manager write
    by set="this/manager/secretary & user" write
    by self write
    by * read
```

La syntaxe précise de ce type de déclaration est assez complexe, et de plus n'est documentée nulle part à l'heure actuelle. L'opérateur & agit comme un opérateur de comparaison entre le membre gauche, qui s'interprète comme une traversée de l'arbre à partir de l'entrée affectée, et le membre droit, qui s'interprète comme l'utilisateur courant. L'expressivité de cette construction est très grande, et permet de mettre en place des politiques d'accès très fines.

Dans le cas ci-dessous, les utilisateurs q et moneypenny peuvent tous les deux modifier les attributs de l'utilisateur james :

```
# l'utilisateur
dn: uid=james,ou=users,dc=domain,dc=tld
objectClass: inetOrgPerson
uid: james
manager: uid=q,ou=users,dc=domain,dc=tld
```

```
# son responsable
dn: uid=q,ou=users,dc=domain,dc=tld
objectClass: inetOrgPerson
uid: q
secretary: uid=moneypenny,ou=users,dc=domain,dc=tld

# la secrétaire de celui-ci
dn: uid=moneypenny,ou=users,dc=domain,dc=tld
objectClass: inetOrgPerson
uid: moneypenny
```

3 Gestion des mots de passe

Après avoir présenté plusieurs recettes concernant la protection des mots de passes, en voici d'autre concernant leur gestion : qualité, expiration et synchronisation.

3.1 Usage

Laisser les utilisateurs gérer leur propre mot de passe, c'est bien. Vérifier qu'ils n'utilisent pas n'importe quel mot de passe, c'est encore mieux. En effet, la vulnérabilité aux attaques par dictionnaire ou par force brute est une affaire de sécurité importante, en particulier dans le cas d'un annuaire LDAP utilisé comme système d'authentification centralisé. Mettre en place une politique de gestion des mots de passe permet de définir un certain nombre de critères de sécurité concernant d'une part leur renouvellement, afin de rejeter ceux jugés trop faibles, mais également leur emploi, afin de se protéger contre ces formes d'attaque.

Cette fonctionnalité est implémentée dans OpenLDAP par le greffon **ppolicy**. Une politique est définie par une entrée de la classe **pwdPolicy**, dont chacun des attributs gouverne un des critères de cette politique. Il est possible d'attribuer une politique spécifique à chaque entrée de l'annuaire, ainsi que de définir une politique par défaut qui s'appliquera à tous.

L'exemple ci-dessous montre ainsi la définition de deux politiques différentes, la première étant la politique par défaut, la seconde beaucoup plus restrictives étant appliquée à certains comptes utilisateurs seulement.

Voici d'abord la partie concernant la configuration de slapd :

```
# chargement des schémas
include /usr/share/openldap/schema/ppolicy.schema
...
# chargement des modules
moduleload ppolicy.la
...
# politique de mots de passes
```

```
overlay ppolicy
ppolicy_default cn=laxist,ou=policies,dc=domain,dc=tld
```

Puis les données de l'annuaire :

```
# politique laxiste:
# - pas d'expiration
# - pas de longueur minimale
# - pas d'historique
# - pas de vérification de qualité
# - pas de verrouillage
dn: cn=laxist,ou=policies,dc=domain,dc=tld
cn: laxist
objectClass: pwdPolicy
objectClass: organizationalRole
pwdAttribute: userPassword
pwdMaxAge: 0
pwdInHistory: 0
pwdCheckQuality: 0
pwdLockout: FALSE

# politique sécuritaire
# - expiration au bout d'une semaine
# - longueur minimale fixée à 8 caractères
# - historique des 4 mots de passes précédent
# - vérification stricte de la qualité
# - verrouillage permanent après 3 tentatives
dn: cn=fascist,ou=policies,dc=domain,dc=tld
cn: fascist
objectClass: pwdPolicy
objectClass: organizationalRole
pwdAttribute: userPassword
pwdMaxAge: 604800
pwdMinLength: 8
pwdInHistory: 4
pwdCheckQuality: 2
pwdLockout: TRUE
pwdMaxFailure: 3
pwdLockoutDuration: 0

# cet utilisateur est soumis à la politique par défaut
dn: uid=foo,ou=users,dc=domain,dc=tld
objectClass: posixAccount
objectClass: person
uid: foo
cn: foo
sn: foo
```

```
gidNumber: 5000
gidNumber: 5000
homeDirectory: /home/foo
userPassword: oldpassword
```

```
# cet utilisateur est soumis à la politique contraignante
dn: uid=bar,ou=users,dc=domain,dc=tld
objectClass: posixAccount
objectClass: person
uid: bar
cn: bar
sn: bar
gidNumber: 5001
gidNumber: 5000
homeDirectory: /home/bar
userPassword: oldpassword
pwdPolicySubentry: cn=fascist,ou=policies,dc=domain,dc=tld
```

Ces données en place, voici ce que donne le changement des mots de passe pour chacun des deux utilisateurs ;

```
[guillaume@oberkampf ~ articles]$ ldappasswd -x -D uid=foo,ou=users,dc=domain,dc=tld -w oldp
[guillaume@oberkampf ~ articles]$ ldappasswd -x -D uid=bar,ou=users,dc=domain,dc=tld -w oldp
Result: Constraint violation (19)
Additional info: Password fails quality checking policy
```

Et une fois les mots de passe changés, voici ce que donne un retour au mot de passe initial, encore une fois pour les deux utilisateurs :

```
[guillaume@oberkampf ~]$ ldappasswd -x -D uid=foo,ou=users,dc=domain,dc=tld -w new -s oldpas
[guillaume@oberkampf ~]$ ldappasswd -x -D uid=bar,ou=users,dc=domain,dc=tld -w password -s o
Result: Constraint violation (19)
Additional info: Password is in history of old passwords
```

Dans les deux cas, la politique libérale valide les changements, tandis que la politique restrictive les rejette.

A noter que ce greffon implémente une proposition de l'IETF, "Password Policy for LDAP Directories", datant de 2005, il y a donc de fortes chances qu'il devienne rapidement une fonctionnalité standard des annuaires LDAP.

Pour plus d'information, consulter :

- la page de manuel `slapo-ppolicy(5)`
- la page greffon du guide d'administration d'OpenLDAP³
- le texte de la proposition de l'IETF⁴
- l'article "Quand LDAP rencontre Tally", dans GLMF n° 109

3. <http://www.openldap.org/doc/admin24>

4. <http://tools.ietf.org/draft/draft-behera-ldap-password-policy/draft-behera-ldap-password-policy-09.txt>

3.2 Qualité

La politique mise en place à la recette précédente impose une longueur minimale, mais aucune robustesse : 'aaaaaaaa' est ainsi un mot de passe valide au regard de cette politique. Il est donc nécessaire de la compléter pour obtenir un niveau de sécurité suffisant.

Le greffon **ppolicy** ne permet pas de vérifier la qualité des mots de passe directement, mais permet de d'externaliser cette tâche. L'attribut **pwdCheckModule** définit un greffon à utiliser pour cette tâche. Il n'existe pas d'implémentation de ce module dans OpenLDAP, il faut soit le développer soi-même (c'est relativement trivial, et expliqué dans la page de manuel), soit utiliser une implémentation réalisée par un tiers, comme par exemple `check_password`⁵.

La mise en place de ce module consiste à le compiler, puis à l'installer avec les autres greffons, dans `/usr/lib64/openldap`. Il est également disponible sous mandriva dans le paquetage `openldap-check_password`. Il faut ensuite définir une qualité minimale dans son fichier de configuration `/etc/openldap/check_password.conf`. Cette qualité se définit comme le nombre de classes de caractères utilisés, ces classes étant les minuscules, les majuscules, les chiffres et la ponctuation. Il faut ensuite étendre la définition de la politique initiale, en rajoutant notamment la classe **pwdPolicyChecker** :

```
# politique toujours plus sécuritaire
# - même contraintes que précédemment
# - vérification de la qualité du mot de passe
dn: cn=morefascist,ou=policies,dc=domain,dc=tld
cn: morefascist
objectClass: pwdPolicy
objectClass: pwdPolicyChecker
objectClass: organizationalRole
pwdAttribute: userPassword
pwdMaxAge: 604800
pwdMinLength: 8
pwdInHistory: 4
pwdCheckQuality: 2
pwdLockout: TRUE
pwdMaxFailure: 3
pwdLockoutDuration: 0
pwdCheckModule: check_password.so
```

Le mot de passe qui était accepté jusqu'à présent est maintenant refusé :

```
[guillaume@oberkamp ~]$ ldappasswd -x -D uid=foo,ou=users,dc=domain,dc=tld -w oldpassword -
Result: Constraint violation (19)
Additional info: Password fails quality checking policy
```

Pour plus d'information, consulter :

5. <http://open.calivia.com/projects/openldap>

- la page de manuel `slapo-ppolicy(5)`
- la page greffon du guide d'administration d'OpenLDAP ⁶
- l'article "Quand LDAP rencontre Tally", dans GLMF n° 109
- la documentation du module `check_password`

3.3 Expiration

La classe **shadowAccount** permet d'apporter à la gestion des comptes Unix les mêmes fonctionnalités que les `shadow passwords`. Et en particulier, de gérer leur expiration à une date fixée. Néanmoins, cette expiration est en fait gérée par un client particulier (`pam_ldap`), et ne concerne donc que les comptes shell. Tout autre forme d'utilisation du compte par une autre client, comme une authentification par `mod_ldap`, par exemple, n'est pas affectée.

Le greffon **ppolicy**, toujours lui, apporte une solution plus générale, gérée par le serveur. Lorsque l'attribut **pwdLockout** est vraie, un nombre d'échecs d'authentification supérieur à la valeur de l'attribut **pwdMaxFailure** entraîne le verrouillage du compte pour une durée spécifiée par l'attribut **pwdLockout-Duration**. C'est l'attribut opérationnel **pwdAccountLockedTime** qui garde une trace de moment où ce compte est verrouillé. En donnant la valeur `000001010000Z` à cet attribut, le compte est immédiatement et définitivement verrouillé.

Voici un exemple qui montre la mise en place de ce verrouillage :

```
[guillaume@oberkamp]$ ldapsearch -x -D uid=bar,ou=users,dc=domain,dc=tld -w password
...
[guillaume@oberkamp ~]$ ldapmodify -x -D cn=admin,ou=roles,dc=domain,dc=tld -w password <<I
> dn: uid=bar,ou=users,dc=domain,dc=tld
> changetype: modify
> add: pwdAccountLockedTime
> pwdAccountLockedTime: 000001010000Z
modifying entry "uid=bar,ou=users,dc=domain,dc=tld"
[guillaume@oberkamp ~]$ ldapsearch -x -D uid=bar,ou=users,dc=domain,dc=tld -w password
ldap_bind: Invalid credentials (49)
```

Malheureusement, il ne semble pas possible d'utiliser cet attribut pour planifier l'invalidation d'un compte à l'avance. Même en utilisant une date située dans le futur, le verrouillage est immédiat. La seule différence entre l'utilisation d'une valeur arbitraire et la valeur `000001010000Z` semble être que dans le premier cas, l'annuaire supprime l'attribut lorsque la durée d'invalidation est passée.

Mis à part cette fonctionnalité manquante, l'utilisation de **ppolicy** n'est pas censé compléter celle de la classe **shadowAccount**, mais la remplacer complètement. En effet, `pam_ldap` est capable d'interpréter les codes d'erreurs supplémentaires renvoyés par le serveur (à condition que la directive `ppolicy_use_lockout` autorise ceux-ci) pour indiquer à l'utilisateur les raisons exactes de l'échec d'authentification.

6. <http://www.openldap.org/doc/admin24>

Cette invalidation de compte reste néanmoins limitée aux opérations d'authentification. Le compte existe toujours vis-à-vis de n'importe quelle requête de sélection des comptes utilisateurs, comme par exemple une liste de diffusion dont les membres sont gérés dynamiquement.

Pour plus d'information, consulter :

- la page de manuel `slapo-ppolicy(5)`
- la page greffon du guide d'administration d'OpenLDAP ⁷
- le texte de la proposition de l'IETF ⁸
- l'article "Quand LDAP rencontre Tally", dans GLMF n° 109

3.4 Synchronisation

En utilisant OpenLDAP comme solution de stockage pour Samba, il devient très facile d'étendre sa base de comptes Unix à un domaine Windows, ce qui évite notamment les problèmes de synchronisation entre bases de comptes différentes. Néanmoins, dans le monde Windows, le client ne délègue pas l'authentification au contrôleur de domaine, mais effectue la comparaison du mot de passe de l'utilisateur lui-même (ce qui permet notamment de s'authentifier ensuite localement sans être connecté). Ceci implique que les mots de passes soient stockés sous une forme exploitable par le client en question, ce qui n'est pas le cas des différents formats possibles pour OpenLDAP, et varie également en fonction des versions de Windows... En conséquence, le mot de passe d'un utilisateur va être stocké plusieurs fois, avec avec des méthodes de chiffage différentes, dans des attributs dédiés : **userPassword**, **sambaLMPassword** et **sambaNTPassword**. Dès lors que l'information est dupliquée, comment assurer que ces différents attributs correspondent toujours au même mot de passe ?

Une solution ad-hoc consiste à mettre en place une procédure dédiée pour changer ce mot de passe, par exemple avec un script qui assure lui-même le chiffage du mot de passe et la modification des différents attributs simultanément. Et à s'assurer également que les utilisateurs ne peuvent pas aller changer individuellement ceux-ci directement dans l'annuaire... Une opération délicate à mettre en place, et qui touche à un sujet sensible également en matière de sécurité. Bref, pas vraiment l'idéal.

Une solution beaucoup plus simple et élégante vient du greffon **smbk5pwd**. Celui-ci a pour fonction de modifier l'opération de changement du mot de passe au niveau du serveur pour automatiquement mettre à jour l'ensemble des attributs concernés.

Voici la configuration nécessaire au niveau du serveur :

```
# chargement des schémas
include /usr/share/openldap/schema/samba.schema
...
# chargement des modules
```

7. <http://www.openldap.org/doc/admin24>

8. <http://tools.ietf.org/draft/draft-behera-ldap-password-policy/draft-behera-ldap-password-policy-09.txt>


```
moduleload smb5pwd.la
```

```
...
```

```
# synchronisation des mots de passe
```

```
overlay smb5pwd
```

L'exemple suivant montre l'entrée correspondant à un utilisateur avant et après un changement de mots de passe. Les 3 attributs ont bien été modifiés, ainsi que l'attribut **sambaPwdLastSet**.

```
[guillaume@oberkampf ~]$ ldapsearch -x -D uid=foo,ou=users,dc=domain,dc=tld -w oldpassword -
```

```
dn: uid=foo,ou=users,dc=domain,dc=tld
```

```
objectClass: posixAccount
```

```
objectClass: inetOrgPerson
```

```
objectClass: sambaSamAccount
```

```
uid: foo
```

```
cn: foo
```

```
sn: foo
```

```
gidNumber: 5000
```

```
gidNumber: 5000
```

```
homeDirectory: /home/foo
```

```
sambaSID: S-1-5-21-2709371413-4020681702-788637496-5012
```

```
userPassword:: e1NTSEF90DBhdmFQMjArVDQ5M1dlldUIyelIvOC9TT3RYa0J3UGU=
```

```
sambaPwdLastSet: 1205791566
```

```
sambaLMPassword: c9b81d939d6fd80cd408e6b105741864
```

```
sambaNTPassword: bb330a886fd4c711a0a3f42d637756d7
```

```
...
```

```
[guillaume@oberkampf ~]$ ldappasswd -x -D uid=foo,ou=users,dc=domain,dc=tld -w oldpassword -
```

```
[guillaume@oberkampf ~]$ ldapsearch -x -D uid=foo,ou=users,dc=domain,dc=tld -w newpassword -
```

```
dn: uid=foo,ou=users,dc=domain,dc=tld
```

```
objectClass: posixAccount
```

```
objectClass: inetOrgPerson
```

```
objectClass: sambaSamAccount
```

```
uid: foo
```

```
cn: foo
```

```
sn: foo
```

```
gidNumber: 5000
```

```
gidNumber: 5000
```

```
homeDirectory: /home/foo
```

```
sambaSID: S-1-5-21-2709371413-4020681702-788637496-5012
```

```
userPassword:: e1NTSEF9ZTdCWHDraXB4MEJzM1l1YUUzMHM5UmhJNVg3NVN5eEk=
```

```
sambaPwdLastSet: 1205791753
```

```
sambaLMPassword: 09eeab5aa415d6e4d408e6b105741864
```

```
sambaNTPassword: bbcef4ffcfce931235927d4134505691b
```

```
...
```

Il faut également imposer à Samba d'utiliser l'opération étendue de changement de mot de passe (ExOp PasswordChange) plutôt que la manipulation directe des attributs concernés, dans le fichier `/etc/samba/smb.conf` :

```
ldap password sync = only
```

Au passage, cette configuration règle le problème soulevé dans l'article "Quand LDAP rencontre Tally", dans GLMF n° 109, à savoir que l'attribut **pwdAttribute** est ignoré par le greffon **ppolicy**, et qu'il est donc impossible d'imposer une politique de mots de passes aux mots de passe Windows.

Ce greffon gère également les clés kerberos, lorsque l'annuaire LDAP est utilisé pour stocker les principaux d'un KDC Heimdal.

Attention, ce greffon ne fait pas partie des greffons standard d'OpenLDAP, il est nécessaire de le compiler manuellement. Les sources sont disponibles dans le répertoire **contrib/slapd-modules/smbk5pwd**. Il est également disponible dans le paquetage **openldap-smbk5pwd** sous Mandriva.

Pour plus d'information, consulter :

- le fichier README présent dans le répertoire des sources du greffon **smb5pwd**

3.5 Externalisation

La recette précédente montrait un exemple d'intégration, en fusionnant les informations d'une autre base d'utilisateurs, celle de Samba, au sein de l'annuaire, et en assurant la synchronisation des mots de passe quand ceux-ci changeaient. Néanmoins, cette stratégie n'est pas toujours possible à mettre en place. Comment faire alors pour éviter le recours à des systèmes de synchronisations ad-hoc ?

Une solution possible dans ce cas est l'inverse de la solution précédente : au lieu d'intégrer, on va externaliser le processus. Toute tentative d'authentification sur l'annuaire sera alors réalisée par le biais d'un des nombreux mécanismes supportés par SASL sur un système distinct.

La mise en place de SASL pourrait à elle seule faire l'objet d'un article dédié. L'exemple utilisé ici utilise un autre annuaire, par exemple un serveur Active Directory. Le fichier de configuration **saslauthd.conf** ressemble à ceci :

```
ldap_servers: ldaps://dc.domain.tld/

ldap_search_base: cn=users,dc=ad,dc=domain,dc=tld
ldap_filter: (userPrincipalName=%u)

ldap_bind_dn: cn=saslauthd,cn=users,dc=ad,dc=domain,dc=tld
ldap_password: secret
```

Le daemon **saslauthd** est lancé de façon à utiliser le mécanisme LDAP, et à combiner le nom du royaume SASL avec le nom d'utilisateur :

```
saslauthd -a ldap -r
```

Lorsque **saslauthd** reçoit une demande d'authentification pour **utilisateur@royaume**, il effectue une connexion vers l'annuaire Active Directory sous l'identité **cn=saslauthd,cn=users,dc=ad,dc=domain,dc=tld**, recherche un

utilisateur correspondant à la requête **userPrincipalName=utilisateur@royaume**, puis essaye de s'authentifier sur cet annuaire avec cette identité, et le mot de passe fourni.

Il est possible de vérifier la validité de l'installation avec le programme **testsaslauthd**.

```
testsaslauthd -u utilisateur@royaume -p password
testsaslauthd -u utilisateur@royaume -p wrongpassword
```

Une fois SASL en place, il faut configurer OpenLDAP pour l'utiliser, par le biais d'un fichier de configuration **slapd.conf**, à placer dans le répertoire de configuration de SASL, **/etc/sasl2** sur une distribution Linux. Ce fichier précise quels protocoles sont susceptibles d'être utilisés, et par quelle méthode. Dans le cas de notre exemple, ce fichier ressemble à ceci :

```
mech_list: plain
pwcheck_method: saslauthd
saslauthd_path: /var/run/sasl2/mux
```

Enfin, ce mécanisme d'externalisation étant sélectif, il reste à remplacer pour les utilisateurs dont on souhaite externaliser l'authentification la valeur de l'attribut **userPassword** par la chaîne **{SASL}utilisateur@royaume**. Ainsi, il est tout à fait possible de n'externaliser que certaines authentifications (typiquement, les utilisateurs), tout en continuant d'assurer en interne les autres (typiquement, les comptes administratifs).

Le mécanisme est alors en place et fonctionne. Il y a cependant une limitation. Lorsque l'utilisateur effectue un changement de mot de passe, OpenLDAP transmet cette opération via SASL également. Mais ceci ne fonctionne que si la session est également authentifiée par SASL, par opposition au mécanisme d'authentification dit simple d'OpenLDAP (voire la recette 9.1), et uniquement pour l'utilisateur en cours. Autrement dit, un administrateur ne peut pas changer ainsi le mot de passe d'un autre utilisateur.

Pour plus d'information, consulter :

- la page sécurité du guide d'administration d'OpenLDAP⁹

4 Gestion des groupes

Que ce soit au niveau du système Unix, ou au niveau de l'annuaire LDAP, les groupes forment la base d'une politique d'autorisation lisible. Ils sont donc incontournables, mais leur mise en oeuvre n'est pas totalement triviale. Les relations entre groupes système et groupes LDAP, d'une part, ainsi que la façon dont est définie l'appartenance d'un utilisateur à un groupe, d'autre part, recèlent quelques subtilités.

Les recettes suivantes montrent comment unifier la définition des groupes entre OpenLDAP et le système sous-jacent, puis comment utiliser les définitions dynamiques de groupes pour faciliter la gestion de ceux-ci.

9. <http://www.openldap.org/doc/admin24>

4.1 Représentation des groupes POSIX

Ceci n'est pas vraiment une recette, mais plutôt une clarification de la représentation des groupes système au sein d'un annuaire LDAP, nécessaire à la compréhension des recettes suivantes.

Dans le modèle POSIX, un utilisateur appartient forcément à un groupe, dit groupe primaire, et éventuellement à plusieurs autres groupes, dit secondaires. Dans le cas d'utilisation de fichiers plats classiques, la référence (gid) du groupe primaire est consignée dans la définition de l'utilisateur (`/etc/passwd`), tandis que l'appartenance éventuelle à d'autres groupes est consignée par la mention du nom de l'utilisateur dans la définition de ces groupes (fichier `/etc/group`). Autrement dit, l'utilisateur pointe vers son groupe primaire, tandis que les groupes pointent vers les utilisateurs dont ils sont un groupe secondaire. Et malgré qu'elle puisse s'exprimer d'une façon ou d'une autre, l'information d'appartenance à un groupe n'est jamais dupliquée, empêchant toute incohérence.

Avec les données suivantes :

```
[guillaume@oberkampf ~]$ cat /etc/passwd
...
foo:x:5000:5000:user foo:/home/foo:/bin/bash
bar:x:5001:5000:user bar:/home/bar:/bin/bash
baz:x:5002:5001:user baz:/home/baz:/bin/bash

[guillaume@oberkampf ~]$ cat /etc/group
...
admins:x:5000:
users:x:5001:foo,bar
```

On obtient les résultats suivants :

```
[guillaume@oberkampf ~]$ id foo
uid=5000(foo) gid=5000(admins) groupes=5000(admins),5001(users)
[guillaume@oberkampf ~]$ id bar
uid=5001(bar) gid=5000(admins) groupes=5000(admins),5001(users)
[guillaume@oberkampf ~]$ id baz
uid=5002(baz) gid=5000(users) groupes=5001(users)
[guillaume@oberkampf ~]$ getent group admins
admins:x:5000:
[guillaume@oberkampf ~]$ getent group users
users:x:5001:foo,bar
```

Le résultat de la commande `getent` est surprenant au premier abord, et la sobriété de sa page de manuel n'aide guère à comprendre, mais en fait cette commande ne fait qu'interroger le fichier `/etc/group` (ou plutôt, la base de données des groupes accessible via `nss`), qui ne contient qu'une partie de l'information. En aucun cas, ce résultat ne doit être interprété comme la liste exhaustive des membres d'un groupe.

Lorsque l'on passe des fichiers plats à un annuaire LDAP, le même principe prévaut : la classe **posixAccount** comporte un attribut **gidNumber**, pour référence le groupe primaire, et la classe **posixGroup** comporte un attribut multivalué **memberUid** pour référencer les utilisateurs pour lequel ce groupe est un groupe secondaire.

```
dn: uid=foo,ou=users,dc=domain,dc=tld
objectClass: posixAccount
objectClass: person
uid: foo
cn: foo
sn: foo
uidNumber: 5000
gidNumber: 5000
homeDirectory: /home/foo
loginShell: /bin/bash
```

```
dn: uid=bar,ou=users,dc=domain,dc=tld
objectClass: posixAccount
objectClass: person
uid: bar
cn: bar
sn: bar
uidNumber: 5001
gidNumber: 5000
homeDirectory: /home/bar
loginShell: /bin/bash
```

```
dn: uid=baz,ou=users,dc=domain,dc=tld
objectClass: posixAccount
objectClass: person
uid: baz
cn: baz
sn: baz
uidNumber: 5002
gidNumber: 5001
homeDirectory: /home/baz
loginShell: /bin/bash
```

```
dn: cn=admins,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
cn: admins
gidNumber: 5000
```

```
dn: cn=users,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
```

```
cn: users
gidNumber: 5001
memberUid: foo
memberUid: bar
```

Et les résultats obtenus sont strictement les mêmes que précédemment :

```
[guillaume@oberkampf ~]$ id foo
uid=5000(foo) gid=5000(admins) groupes=5000(admins),5001(users)
[guillaume@oberkampf ~]$ id bar
uid=5001(bar) gid=5000(admins) groupes=5000(admins),5001(users)
[guillaume@oberkampf ~]$ id baz
uid=5002(baz) gid=5000(users) groupes=5001(users)
[guillaume@oberkampf ~]$ getent group admins
admins:x:5000:
[guillaume@oberkampf ~]$ getent group users
users:x:5001:foo,bar
```

Moralité, il n'est absolument pas nécessaire d'inclure les utilisateurs dont c'est le groupe primaire parmi les membres d'un groupe, du point de vue du système sous-jacent. Néanmoins, pour bénéficier de certaines fonctionnalités de LDAP liées aux groupes, comme par exemple la recette 2.1, ou encore la recette 6.1, il est nécessaire que cette relation d'appartenance soit également identifiée par l'annuaire LDAP.

La première solution, c'est de dupliquer cette information, en constituant donc un groupe LDAP parallèle au groupe Posix.

```
dn: cn=admins,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
cn: admins
gidNumber: 5000

dn: cn=ldap_admins,ou=groups,dc=domain,dc=tld
objectClass: groupOfNames
cn: ldap_admins
member: uid=foo,ou=users,dc=domain,dc=tld
member: uid=bar,ou=users,dc=domain,dc=tld
```

Cette solution présente un double problème de duplication d'information. D'abord, le deuxième groupe constitue un doublon du premier, dans la mesure où il sert juste à exprimer la même chose, mais différemment. Ensuite, la relation d'appartenance des utilisateurs à ce groupe est clairement dupliquée, au niveau de leur attribut **gidNumber** d'une part, de l'attribut **member** du deuxième groupe d'autre part. Et dès lors qu'il y a duplication, il y a risque d'introduction d'incohérence. Les deux recettes suivantes présentent des solutions à ces problèmes.

4.2 Fusion des groupes POSIX et LDAP

La recette précédente montrait que l'utilisation de groupes de type **groupOfNames**, pour représenter la relation d'appartenance à un groupe au niveau de l'annuaire LDAP, présentait un problème de redondance avec les groupes de type **posixGroup**. Pour faire disparaître ce problème, il serait intéressant de fusionner ces entrées multiples. Néanmoins, avec le schéma classiquement utilisé (celui décrit par la RFC 2307), ceci n'est pas possible. En effet, les classes **posixGroup** et **groupOfNames** sont toutes deux des classes structurelles, donc impossible à utiliser simultanément pour une même entrée.

La solution consiste à utiliser un schéma alternatif, celui de la RFC 2307bis, une amélioration de la RFC originale qui n'a cependant jamais été finalisée. Dans ce schéma, la classe **posixGroup** est une classe auxiliaire, de la même façon d'ailleurs que la classe **posixAccount**, ce qui permet de la combiner avec une autre classe, et notamment **groupOfNames**. Il devient donc possible de fusionner les deux groupes de l'exemple précédent en un seul :

```
dn: cn=admins,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfNames
cn: admins
gidNumber: 5000
member: uid=foo,ou=users,dc=domain,dc=tld
member: uid=bar,ou=users,dc=domain,dc=tld
```

Dans la configuration du serveur, il faut remplacer les schémas NIS et autofs par le nouveau schéma :

```
# chargement des schémas
include /usr/share/openldap/schema/rfc2307bis.schema
```

Enfin, la configuration `nss_ldap` nécessite quelques ajustements mineurs :

```
nss_schema rfc2307bis
nss_map_attribute uniqueMember member
```

Attention, la classe **groupOfNames** a pour contrainte de posséder au moins un membre. La nature de ce membre étant sans importance, l'utilisation d'un objet dédié de classe **organizationalRole** suffit à satisfaire cette exigence si nécessaire :

```
# utilisateur fantôme
dn: cn=default,ou=roles,dc=domain,dc=tld
description: default group member
cn: default
objectClass: organizationalRole

# groupe nouvellement créé
dn: cn=empty,ou=groups,dc=domain,dc=tld
```

```
objectClass: posixGroup
objectClass: groupOfNames
cn: empty
gidNumber: 5000
member: uid=default,ou=roles,dc=domain,dc=tld
```

La migration entre les deux schémas n'est pas triviale. En effet, ils définissent les mêmes classes, et sont donc antagonistes, rendant impossible leur coexistence au sein du serveur. La solution la plus simple consiste donc à exporter les données de l'annuaire au format LDIF, à modifier ces données, puis à les réimporter dans l'annuaire après l'avoir vidé, et modifié sa configuration.

Pour plus d'information, consulter :

- la RFC 23707 bis¹⁰

4.3 Groupes dynamiques

La recette précédente règle un des problèmes de redondance, mais pas le deuxième : un utilisateur pointe vers son groupe primaire via son attribut **gidNumber**, et son groupe primaire pointe également vers lui via son attribut **member**. Si les deux valeurs ne sont pas cohérentes, le système et l'annuaire auront des avis divergent sur l'appartenance de cet utilisateur à ce groupe. Comment rendre la situation plus robuste ?

La solution consiste à utiliser le greffon **dynlist**, et à construire le groupe primaire de façon dynamique. Ceci se fait en changeant la définition de ce groupe, d'abord en le passant dans la classe **groupOfURLs** à la place de **groupOfNames**, puis en remplaçant son attribut **member** par un attribut **memberURL**, dont la valeur est une URL. C'est cette URL qui sera évaluée lors de chaque requête de façon à produire le résultat attendu, à savoir la liste des membres du groupe. La classe de l'exemple précédent devient ainsi :

```
dn: cn=admins,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfURLs
cn: admins
gidNumber: 5000
memberURL: ldap:///ou=users,dc=domain,dc=tld??sub?(gidNumber=5000)
```

Coté serveur, Il faut changer la configuration de slapd pour charger le greffon, ainsi qu'un nouveau schéma. Il faut également configurer le greffon lui-même, en indiquant quelle classe et quel attribut de cette classe doivent être évalués dynamiquement. Enfin, le troisième paramètre optionnel précise si cette requête est censée produire des attributs qu'il faut rajouter à l'entrée dynamique, ou plutôt une liste d'entrées dont il faut rajouter les DN dans un nouvel attribut. C'est le cas ici, la requête extrayant l'ensemble des utilisateurs dont l'attribut **gidNumber** vaut 5000, dont il ne faut retenir que les DN, qui doivent constituer

10. <http://www.padl.com/~lukeh/rfc2307bis.txt>

les valeur de l'attribut **member** du groupe dynamique. Le résultat ressemble à ceci :

```
# chargement des schemas
include /usr/share/openldap/schema/dyngroup.schema
...
# chargement des modules
moduleload dynlist.so
...
overlay dynlist
dynlist-attrset groupOfURLs memberURL member
```

Une requête portant sur le groupe déclenche maintenant l'expansion de l'URL, et l'apparition d'un attribut **member** dynamique, comprenant la liste des membres de ce groupe :

```
[guillaume@oberkamp ~]$ ldapsearch -LLL -x cn=admins
dn: cn=admins,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfURLs
gidNumber: 5000
memberURL: ldap:///ou=users,dc=domain,dc=tld??sub?(gidNumber=5000)
cn: admins
member: uid=foo,ou=users,dc=domain,dc=tld
member: uid=bar,ou=users,dc=domain,dc=tld
```

Il est également possible d'empêcher l'expansion dynamique, de façon à inspecter le contenu réel de l'entrée, en utilisant le contrôle **manageDSAit** :

```
[guillaume@oberkamp ~]$ ldapsearch -LLL -x -M cn=admins
dn: cn=admins,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfURLs
gidNumber: 5000
memberURL: ldap:///ou=users,dc=domain,dc=tld??sub?(gidNumber=5000)
cn: admins
```

Il devient maintenant possible de mettre en place l'ACL de groupe décrite à la recette 2.1, à un détail syntaxique près : comme il s'agit d'un groupe dynamique, et non pas d'un groupe statique basé sur la classe **groupOfNames**, il faut préciser le nom de la classe et le nom de l'attribut à évaluer. Le résultat ressemble à ceci :

```
access to dn.subtree="dc=domain,dc=tld"
  by group/groupOfURLs/memberURL="cn=admins,ou=groups,dc=domain,dc=tld write
  by * break
```

Pour plus d'information, consulter :

- la page de manuel `slapo-dynlist(5)`
- la page greffon du guide d'administration d'OpenLDAP ¹¹

4.4 Groupes dynamiques rémanants

La recette précédente montre comment construire des groupes dynamiques, valides aussi bien au niveau système qu'au niveau de l'annuaire LDAP, en se basant sur la relation d'appartenance au groupe primaire. Il est tentant de généraliser ce principe, en définissant ainsi d'autres groupes dynamiques arbitraires. Par exemple, un groupe incluant tous les utilisateurs de l'annuaire :

```
dn: cn=all,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfURLs
gidNumber: 5002
cn: all
memberURL: ldap:///ou=users,dc=domain,dc=tld??sub?(objectClass=posixAccount)
```

Ceci marche très bien vis à vis des requêtes LDAP :

```
[guillaume@oberkampf ~]$ ldapsearch -LLL -x cn=all member
dn: cn=all,ou=groups,dc=domain,dc=tld
member: uid=foo,ou=users,dc=domain,dc=tld
member: uid=bar,ou=users,dc=domain,dc=tld
member: uid=baz,ou=users,dc=domain,dc=tld
```

Malheureusement, ce n'est pas le cas au niveau du système :

```
[guillaume@oberkampf ~]$ id foo
uid=5000(foo) gid=5000(admins) groupes=5000(admins),5001(users)
[guillaume@oberkampf ~]$ id bar
uid=5001(bar) gid=5000(admins) groupes=5000(admins),5001(users)
[guillaume@oberkampf ~]$ id baz
uid=5002(baz) gid=5001(users) groupes=5001(users)
```

Que se passe-t-il donc ? La réponse se trouve dans le fonctionnement du greffon `dynlist`, décrit dans la page de manuel `slapo-dynlist(5)` : l'évaluation de l'attribut dynamique a lieu après une recherche, et non avant. Autrement dit, toute recherche portant sur la valeur d'un attribut dynamique est vouée à l'échec. Il est possible d'obtenir la liste des membres d'un groupe dynamique, mais pas la liste des groupes dynamiques comportant un membre précis. La requête suivante le montre, puisque seul le groupe statique figure dans le résultat :

```
[guillaume@oberkampf ~]$ ldapsearch -LLL -x member=uid=foo,ou=users,dc=domain,dc=tld dn
dn: cn=users,ou=groups,dc=domain,dc=tld
```

¹¹. <http://www.openldap.org/doc/admin24>

Le greffon **autogroup**, qui combine les fonctionnalités des deux types de groupe, est censé apporter une réponse à ce problème. L'attribut **memberURL** correspond ici à un filtre, qui est évalué à chaque insertion, modification ou effacement d'une entrée de l'annuaire. Si cette entrée correspond au filtre, l'attribut **member** du groupe est alors mis à jour en conséquence. Le groupe se comporte donc comme un groupe statique vis-à-vis des opérations de recherche et de comparaison.

Malheureusement, tous mes essais d'utilisation de ce greffon à l'heure actuelle se sont soldés par un échec cuisant. Le fait qu'il s'agisse d'un greffon de la section des contributions, plutôt que d'un greffon officiel, laisse supposer qu'il n'est sans doute pas aussi testé et éprouvé que les autres.

Pour plus d'information, consulter :

- le fichier README présent dans le répertoire des sources du greffon autogroup

5 Mise en place de vues utilisateur

Beaucoup de clients LDAP sont très restrictifs sur la façon dont ils peuvent interroger un annuaire. En particulier, ceux des systèmes restreints embarqués sur des équipements réseaux, sur des imprimantes, etc... Un Call Manager Cisco, par exemple, permet de synchroniser sa propre base d'utilisateur sur un annuaire LDAP, mais avec des contraintes assez pénibles. Comment donc gérer l'intégration de ces matériels, sans pour autant réorganiser la structure de l'annuaire pour se plier à leurs contraintes ?

Une solution consiste à déporter le travail de manipulation des données vers le serveur, et à répondre différemment en fonction du client qui l'interroge : c'est le principe des vues, bien connu des utilisateurs de SGBD. Les recettes suivantes montrent comment réaliser ce genre de chose avec OpenLDAP, pour filtrer des entrées d'une part, pour échanger des attributs d'autre part.

5.1 Filtrage d'entrées

Le problème le plus courant est celui du client ne permettant pas de filtrer des entrées indésirables. Le Call Manager Cisco, par exemple, importe avidement tous les utilisateurs situés sous sa base de recherche, sans permettre de filtrer par exemple ceux qui sont dépourvus d'un numéro de téléphone. Il faut donc identifier ce client lorsqu'il interroge l'annuaire, puis effectuer ce filtrage à sa place.

Puisqu'il s'agit d'exclure des entrées de la réponse, les ACLs sont parfaitement adaptées. Il suffit en effet d'interdire à ce client l'accès à ces entrées, en appliquant le filtre nécessaire pour les sélectionner, pour toute demande émanant de l'adresse IP ou de l'identifiant de ce client :

```
# filtrage des entrées visibles par le call manager aux seuls utilisateurs
# pourvus d'un numéro de téléphone
access to dn.subtree="ou=users,dc=domain,dc=tld" filter=(!(telephoneNumber=*))
```

```

by dn="cn=callmanager,ou=roles,dc=domain,dc=tld" none
by * break

# filtrage des entrées visibles par le photocopieur aux seuls membres
# des ressources humaines ou des affaires financières
access to dn.subtree="ou=users,dc=domain,dc=tld" filter=(!(|(ou=srh)(ou=saf)))
by anonymous peername.ip=192.168.0.1 none
by * break

```

Il est alors facile de vérifier la différence de résultats entre les deux requêtes, en fonction de l'identifiant utilisé :

```

[guillaume@oberkamp ~]$ ldapsearch -x -LLL objectClass=person telephoneNumber
dn: uid=foo,ou=users,dc=domain,dc=tld
telephoneNumber: 1234567890

```

```

dn: uid=bar,ou=users,dc=domain,dc=tld

```

```

[guillaume@oberkamp ~]$ ldapsearch -x -LLL -D cn=callmanager,ou=roles,dc=domain,dc=tld -w s
dn: uid=foo,ou=users,dc=domain,dc=tld
telephoneNumber: 1234567890

```

Attention lors de la mise en place de telles ACLs à l'impact potentiel sur les autres accès. L'utilisation d'ACLs strictement nominative, comme détaillée dans la recette 2.3, est fortement conseillée pour limiter les risques.

Pour plus d'information, consulter :

- la page de manuel `slapd.access(5)`
- la page contrôle d'accès du guide d'administration d'OpenLDAP ¹²

5.2 Échange d'attributs

Un problème plus complexe se pose lorsque le client ne permet pas de choisir les attributs utilisés. Le Call Manager Cisco, toujours lui, impose l'utilisation de l'attribut **telephoneNumber** pour stocker le numéro de téléphone interne, alors que d'autres applications accèdent également à cet attribut, et s'attendent à y voir figurer le numéro externe. Il ne s'agit donc plus de filter cet attribut, mais d'utiliser en fait deux attributs différents, et à présenter l'un ou l'autre sous un même nom en fonction du client.

Les ACLs ne suffisent plus, il faut mettre en place un échange (mapping) d'attributs, au sein d'un contexte virtuel, à l'aide du module de stockage **relay**. L'utilisation de ce contexte virtuel suffira à identifier le client.

L'exemple suivant montre la mise en place d'un contexte virtuel **ou=telephony,dc=domain,dc=tld**, répliquant la branche utilisateur **ou=users,dc=domain,dc=tld**, et dans lequel l'attribut **telephoneNumber** original est remplacé par l'attribut **homePhone**. Le fait que ce contexte virtuel appartienne au suffixe **dc=domain,dc=tld**

12. <http://www.openldap.org/doc/admin24>

impose de déclarer la base virtuelle avant la base réelle dans la configuration de slapd :

```
# chargement du module
moduleload back_relay.la

# mise en place d'un contexte virtuel, dupliquant la branche utilisateur
database          relay
suffix            ou=telephony,dc=domain,dc=tld
overlay           rwm
rwm-suffixmassage ou=users,dc=domain,dc=tld

# substitution de l'attribut telephoneNumber
rwm-map attribute telephoneNumber homePhone
rwm-map attribute telephoneNumber
```

Cette fois-ci, c'est le contexte de recherche utilisé qui fait la différence de résultats entre les deux requêtes :

```
[guillaume@oberkampf ~]$ ldapsearch -x -LLL -b ou=users,dc=domain,dc=tld objectClass=person
dn: uid=foo,ou=users,dc=domain,dc=tld
telephoneNumber: 1234567890
homePhone: 7890
```

```
[guillaume@oberkampf ~]$ ldapsearch -x -LLL -b ou=telephony,dc=domain,dc=tld objectClass=person
dn: uid=foo,ou=telephony,dc=domain,dc=tld
telephoneNumber: 7890
```

Cette stratégie permet également de mettre en oeuvre le même filtrage des entrées à partir d'ACLs que la précédente, en simplifiant l'identification du client, puisque le contexte dédié le fait implicitement. De plus, les ACLs exotiques sont ainsi isolées des ACLs de la base principale, réduisant les possibilités d'interactions non désirées :

```
access to dn.subtree="ou=telephony,dc=domain,dc=tld" filter=(!(telephoneNumber=*))
by * none
```

Attention également à masquer les attributs sensibles, c'est-à-dire notamment les mots de passe, dans cette nouvelle branche, sous peine d'ouvrir une faille de sécurité remarquable... Le plus simple est de ne garder dans le contexte virtuel que les attributs nécessaires, et d'ignorer les autres :

```
rwm-map attribute telephoneNumber homePhone
rwm-map attribute telephoneNumber
rwm-map attribute *
```

Néanmoins, un bug dans les versions 2.4.11 et antérieures empêche ceci de fonctionner correctement. Il est plus simple dans ce cas d'utiliser des ACLs, en utilisant la syntaxe décrite dans la recette 2.2 pour ne pas avoir à énumérer un par un les attributs à protéger :

```
access to dn.subtree="ou=telephony,dc=domain,dc=tld"
    attrs=@krb5KDCEntry,@krb5Principal,@sambaSamAccount,@posixAccount
    by * none
```

Pour la petite histoire, si cette stratégie fonctionne parfaitement en ligne de commande, un problème de gestion du contrôle `pagedResult` dans le contexte virtuel avec la version actuelle d'OpenLDAP l'empêche de fonctionner avec le Call Manager Cisco qui sert d'exemple dans cette recette. Le problème est connu, et en attente de résolution (ITS #5724).

Pour plus d'information, consulter :

- la page de manuel `slapd-relay(5)`
- la page de manuel `slapo-rwm(5)`
- la page stockage du guide d'administration d'OpenLDAP ¹³
- la page greffons du guide d'administration d'OpenLDAP ¹⁴

6 Cohérence des données

Les systèmes de gestion de bases de données relationnels proposent de façon standardisés de nombreux moyens d'assurer la cohérence des données qu'ils stockent. OpenLDAP propose également certains mécanismes assurant les mêmes fonctionnalités.

6.1 Intégrité référentielle

Dès lors qu'on stocke des références vers des données, il faut s'assurer que lorsque ces données changent, les références concernées changent également. C'est le problème de l'intégrité référentielle. Par exemple, les groupes contiennent des références vers les utilisateurs (dans le schéma RFC2307 bis), et il faut s'assurer que quand ces utilisateurs disparaissent, ou changent d'identifiant, les groupes soient également mis à jour.

OpenLDAP propose un contrôle de l'intégrité référentielle avec le greffon **refint**. Celui-ci assure qu'en cas de changement de DN ou d'effacement d'objet, tous les attributs pointant vers l'objet en question sont automatiquement mis à jour. La contrainte initiale peut se configurer ainsi :

```
# chargement des modules
moduleload refint.so
...
overlay refint
refint_attributes member
```

Les commandes suivantes montrent l'effet de l'effacement de l'utilisateur **bar** sur le groupe **users**, à savoir que cet utilisateur est effacé de la liste des membres de ce groupe :

13. <http://www.openldap.org/doc/admin24>

14. <http://www.openldap.org/doc/admin24>

```
[guillaume@oberkampf ~]$ ldapsearch -x -LLL cn=users
dn: cn=users,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfNames
cn: users
member: uid=bar,ou=users,dc=domain,dc=tld
member: uid=foo,ou=users,dc=domain,dc=tld
gidNumber: 5001
[guillaume@oberkampf ~]$ ldapdelete -x -D cn=admin,ou=roles,dc=domain,dc=tld -w secret uid=bar
[guillaume@oberkampf ~]$ ldapsearch -x -LLL cn=users
dn: cn=users,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfNames
cn: users
member: uid=foo,ou=users,dc=domain,dc=tld
gidNumber: 5001
```

Et les commandes suivantes montrent l'effet du renommage de l'utilisateur **bar** en **baz** sur ce même groupe, à savoir que la liste des membres du groupe est mise à jour :

```
[guillaume@oberkampf ~]$ ldapmodrdn -x -D cn=admin,ou=roles,dc=domain,dc=tld -w secret uid=bar,ou=users,dc=domain,dc=tld,1,uid=baz
[guillaume@oberkampf ~]$ ldapsearch -x -LLL cn=users
dn: cn=users,ou=groups,dc=domain,dc=tld
objectClass: posixGroup
objectClass: groupOfNames
cn: users
gidNumber: 5001
member: uid=baz,ou=users,dc=domain,dc=tld
```

Pour plus d'information, consulter :

- la page de manuel `slapo-refint(5)`
- la page greffon du guide d'administration d'OpenLDAP¹⁵

6.2 Unicité des attributs

Un autre problème de validité classique concerne l'unicité de certaines données. Par exemple, chaque utilisateur doit posséder un identifiant alphanumérique (login) et numérique (uid) distinct. Dans un annuaire LDAP, l'attribut utilisé pour former le RDN d'un objet est déjà couvert par une telle contrainte. L'attribut **uid** de la classe **posixAccount** servant de RDN, il ne peut ainsi y avoir deux utilisateurs de même login dans une même branche LDAP. Si les utilisateurs sont répartis dans plusieurs branches, ceci n'est pas suffisant pour assurer le respect de la contrainte initiale. Et de toute façon, ceci ne couvre pas l'attribut **uidNumber**.

15. <http://www.openldap.org/doc/admin24>

OpenLDAP propose une véritable contrainte d'unicité par le biais du greffon **unique**. Celui-ci permet d'assurer qu'au sein d'une portée donnée, certains attributs aient tous une valeur unique. La contrainte initiale peut donc se configurer ainsi :

```
# chargement des modules
moduleload unique.la
...
overlay unique
unique_base ou=users,dc=domain,dc=tld
unique_attributes uid uidNumber
```

Et toute tentative de modification d'attribut violant cette contrainte se traduit immédiatement par une erreur, comme le montrent les commandes suivantes :

```
[guillaume@oberkampf ~]$ ldapmodify -x -D cn=admin,ou=roles,dc=domain,dc=tld -w password <<I
> dn: uid=bar,ou=users,dc=domain,dc=tld
> changetype: modify
> replace: uidNumber
> gidNumber: 5000
ldap_add: Constraint violation (19)
        additional info: some attributes not unique
```

Pour plus d'information, consulter :

- la page de manuel `slapo-unique(5)`
- la page greffon du guide d'administration d'OpenLDAP ¹⁶

7 Audit

Le serveur `slapd` gère des fichiers journaux (logs), au même titre que n'importe quel serveur. Néanmoins, ceux-ci sont principalement dédiés à la surveillance du fonctionnement du serveur lui-même, de façon à résoudre les éventuels problèmes. Le seul filtrage possible concerne les différents sous-systèmes (requêtes, connexions, filtres, etc...). De plus, à moins d'utiliser une version de `syslog` avancée, telle `syslog-ng` ou `rsyslog`, avec stockage dans une base de donnée relationnelle, ces journaux sont stockés dans des fichiers textes plats, ce qui rend leur exploitation peu aisée. Bref, ils sont peu adaptés à la production d'information synthétique de haut niveau.

7.1 Accès aux données

Le greffon **accesslog** permet d'enregistrer sous forme d'entrées dans une base LDAP l'ensemble des opérations effectuées sur les données d'une autre base. Elles peuvent ainsi être consultées par des requêtes LDAP ordinaire.

16. <http://www.openldap.org/doc/admin24>

La configuration passe par la définition de la base dédiée avant celle à surveiller, puis à définir différents paramètres d'enregistrement. En particulier, la directive **logops** spécifie les différents types d'opérations à enregistrer, soit individuellement (ajout, effacement, comparaison, ...) soit par groupe (lecture, écriture, gestion de session). L'exemple de configuration ci-dessous enregistre toutes les modifications effectuées à la base **dc=domain,dc=tld** dans la base **cn=log** :

```
# chargement des modules
moduleload      accesslog.la
...

# base d'audit
database        bdb
suffix          "cn=log"
rootdn          "cn=root,cn=log"
directory       /var/lib/ldap/log
...

# base principale
database        bdb
suffix          "dc=domain,dc=tld"
directory       /var/lib/ldap/main
...

# enregistrement des modifications réussies
# conservation pendant une semaine
overlay accesslog
logdb cn=log
logops writes
logsuccess TRUE
logpurge 07+00:00 01+00:00
```

Voici l'entrée correspondant au changement du mot de passe d'un utilisateur **uid=user,ou=users,dc=domain,dc=tld**, effectuée par **cn=pam,ou=roles,dc=domain,dc=tld**, c'est à dire l'utilisateur système root à travers pam en utilisant la configuration présentée à la recette 1.4. Les valeurs données sont les nouvelles valeurs, les anciennes n'étant enregistrées que si la directive **logold** est utilisée également.

```
# 20080310195005.000004Z, log
dn: reqStart=20080310195005.000004Z,cn=log
objectClass: auditModify
reqStart: 20080310195005.000004Z
reqEnd: 20080310195005.000005Z
reqType: modify
reqSession: 2
reqAuthzID: cn=pam,ou=roles,dc=domain,dc=tld
```

```

reqDN: uid=user,ou=users,dc=domain,dc=tld
reqResult: 0
reqMod: userPassword:= {MD5}9x2+UmKKP40nerSUGXU1xg==
reqMod: pwdChangedTime:= 20080310195005Z
reqMod: pwdFailureTime:-
reqMod: entryCSN:= 20080310195005Z#000000#00#000000
reqMod: modifiersName:= cn=pam,ou=roles,dc=domain,dc=tld
reqMod: modifyTimestamp:= 20080310195005Z

```

Des informations confidentielles pouvant se trouver dans cette base, comme le montre l'exemple précédent, il convient de restreindre son accès par des ACLs, au même titre que la base principale. Il est intéressant également de noter que l'utilisation de ce greffon permet de faire de la synchronisation différentielle avec syncrepl, et donc de ne transmettre que les modifications entre le serveur maître et le serveur esclave.

Pour plus d'information, consulter :

- la page de manuel `slapo-accesslog(5)`
- la page greffon du guide d'administration d'OpenLDAP ¹⁷

7.2 Statistiques d'utilisation

Le module de stockage **back_monitor** permet lui d'activer les statistiques d'utilisation, là encore sous forme d'objets dans une base LDAP. La configuration est minimale, puisque la base de stockage de ces informations est obligatoirement **cn=monitor**, et qu'il n'y a aucune directive :

```

# chargement des modules
moduleload back_monitor.la
...
database monitor

```

Voici l'entrée correspondant aux opérations d'ouverture de session (bind), indiquant notamment le nombre d'opérations effectuées (attribut **monitorOpInitiated**), depuis la date de mise en place du greffon (attribut **createTimestamp**). Tous les attributs en question sont des attributs opérationnels, il faut prendre garde à les demander explicitement dans la requête.

```

# Bind, Operations, Monitor
dn: cn=Bind,cn=Operations,cn=Monitor
structuralObjectClass: monitorOperation
monitorOpInitiated: 7690
monitorOpCompleted: 7690
creatorsName:
modifiersName:
createTimestamp: 20080310194959Z
modifyTimestamp: 20080310194959Z

```

¹⁷. <http://www.openldap.org/doc/admin24>

```
entryDN: cn=Bind,cn=Operations,cn=Monitor
subschemaSubentry: cn=Subschema
hasSubordinates: FALSE
```

Là encore, des ACLs peuvent donc être nécessaires, bien que la confidentialité de ces informations soit moins évidente que dans le cas précédent, et s'apparente à celles de toutes les données générales de fonctionnement du système.

Pour plus d'information, consulter :

- la page de manuel `slapd-monitor(5)`
- la page stockage du guide d'administration d'OpenLDAP ¹⁸

7.3 Métrologie

Le greffon `monitor`, présenté à la recette précédente, fournit de nombreuses statistiques, mais dont l'exploration manuelle présente peu d'intérêt. D'autres sources de données sont également possibles, comme par exemple les outils de statistique de la base BDB utilisée pour stocker l'annuaire. Reste à trouver comment présenter ces données sous une forme accessible.

Munin ¹⁹ est un outil de métrologie, basé comme de nombreux autres sur `rrd-tools`, présentant de grandes facilités d'auto-configuration. Il est basé sur l'emploi de greffons spécifiques à un type de surveillance, et dans le cas d'OpenLDAP, il en existe deux, récupérables chez un contributeur anonyme ²⁰. Le premier de ce plugin utilise les statistiques fournies par le greffon `monitor`, le deuxième utilise l'outil `db.stat` (paquetage `db42-utils` sur une mandriva 2008.0, `db46-utils` sur les versions suivantes). Les figures 1 à 3 sont plus parlantes qu'un long discours.

Hobbit ²¹ est une autre solution de métrologie similaire. Un greffon écrit par Buchan Milne, `bb-openldap` ²², permet de récupérer les informations de `slapd`. En plus des informations de métrologies à proprement parler, il vérifie également la synchronisation des serveurs secondaires, comme présenté dans la recette 8.2.

8 Réplication

La réplication est le mécanisme permettant de mettre en place un ou plusieurs serveurs secondaires, de façon à améliorer la robustesse du service. L'ancien système `slurpd` est obsolète, il a même complètement disparu de la branche 2.4, et il a été aujourd'hui remplacé par `syncrepl`. Les recettes suivantes montrent sa mise en place, présentent quelques façons de surveiller son fonctionnement, puis comment faire en sorte de propager les changements effectués sur un serveur secondaire vers le serveur principal de façon transparente.

18. <http://www.openldap.org/doc/admin24>

19. <http://munin.projects.linpro.no>

20. http://wiki.ruberg.no/index.php/Munin_LDAP_plugins

21. <http://hobbitmon.sourceforge.net>

22. <http://staff.telkomsa.net/~bgmilne/hobbit>

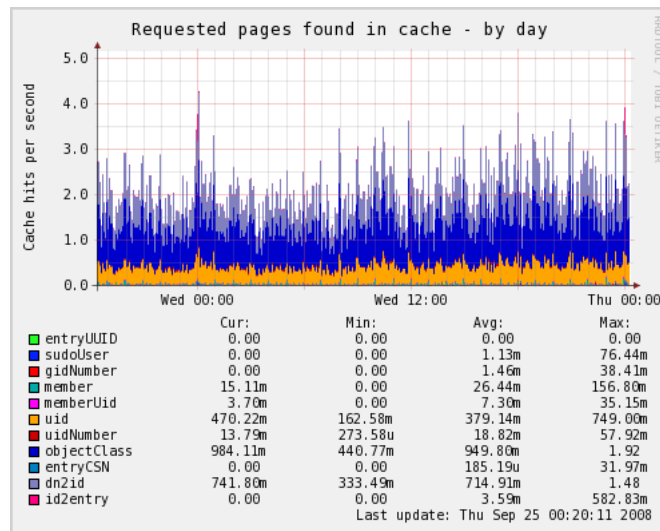


FIGURE 1 – Recherches d'index réussies, par jour

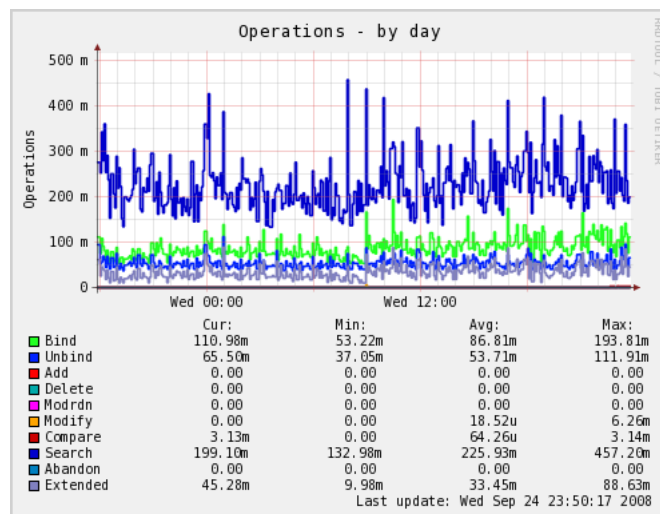


FIGURE 2 – Nombre d'opérations effectuées, par jour

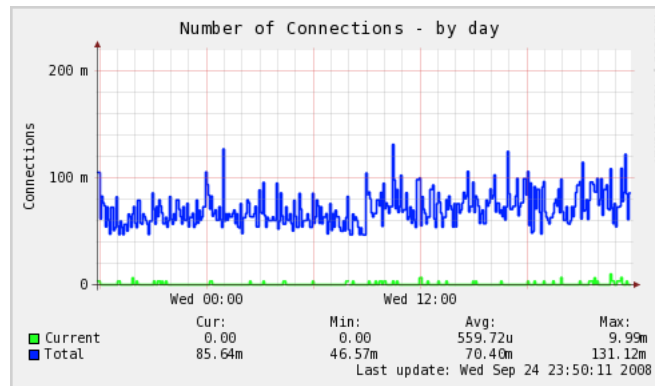


FIGURE 3 – Nombre de connections, par jour

8.1 Mise en place

La mise en place proprement dite ne pose guère de problèmes particuliers. Il s'agit de charger le greffon sur le serveur secondaire (le consommateur), de le configurer proprement, de s'assurer que l'identité utilisée sur le serveur principal (le fournisseur) permet la lecture de tout ce qui doit être répliqué. Le serveur secondaire va alors se mettre à jour de façon automatique. Cependant, quelques précisions peuvent se révéler utiles.

D'abord, il existe deux types de synchronisation : intermittente ou persistante. Dans le premier cas (`refreshOnly`), le consommateur envoie au fournisseur une requête de synchronisation à intervalle régulier. Dans le second cas (`refreshAndPersist`), cette requête persiste au sein du fournisseur, et tout changement sur celui-ci génère automatiquement leur transmission au consommateur. Dans l'exemple suivant, c'est le mode persistant qui est utilisé, comme déterminé par le paramètre `type` :

```
syncrepl rid=123
  provider=ldaps://ldap.domain.com
  type=refreshAndPersist
  searchbase="dc=domain,dc=tld"
  scope=sub
  schemachecking=off
  bindmethod=simple
  binddn="cn=syncrepl,ou=roles,dc=domain,dc=tld"
  credentials=s3cr3t
```

Ceci ne doit pas être confondu avec la synchronisation entière ou différentielle (`delta syncrepl`). La première, la plus simple à mettre en oeuvre transmet au consommateur l'intégralité d'une entrée lorsque celle-ci est modifiée. Ce qui dans le cas des annuaires importants peut générer un trafic important. A l'opposé, la seconde ne transmet que les modifications effectives, ce qui est censé être plus

économique en terme de bande passante. Elle nécessite par contre la mise en place de journaux d'audit, comme détaillé dans la recette 7.1. Il suffit alors de rajouter les paramètres `logbase`, `logfilter` et `syncdata` à l'exemple précédent :

```
syncrepl rid=123
  provider=ldaps://ldap.domain.com
  type=refreshAndPersist
  logbase="cn=log"
  logfilter="(&(objectClass=auditWriteObject)(reqResult=0))"
  syncdata=accesslog
  searchbase="dc=domain,dc=tld"
  scope=sub
  schemachecking=off
  bindmethod=simple
  binddn="cn=syncrepl,ou=roles,dc=domain,dc=tld"
  credentials=s3cr3t
```

Au niveau des permissions, sur le consommateur c'est le `rootdn` qui est utilisé, il n'y a donc besoin d'aucune ACL particulière. Par contre, sur le fournisseur, il est important de vérifier que l'utilisateur utilisé possède toutes les autorisations nécessaires, et n'est pas bridé par de quelconques limites. Ceci à la fois pour la base à répliquer d'une part, et sur la base de log également dans le cas de la réplication différentielle. Une solution radicale est de passer par une ACL dédiée, à placer en tête de liste :

```
# syncrepl: accès en lecture global
access to dn.subtree="dc=domain,dc=tld" by
  dn="cn=syncrepl,ou=roles,dc=domain,dc=tld" read
  by * break

limits dn.exact="cn=syncrepl,ou=roles,dc=domain,dc=tld"
  size.soft=unlimited size.hard=unlimited size.unchecked=unlimited
  time.soft=unlimited time.hard=unlimited
```

Enfin, le paramètre `rid` est un identifiant de réplication interne au consommateur. En d'autres termes, il n'y a pas besoin de s'assurer de son identité entre différents consommateurs.

Pour plus d'information, consulter :

- la page de manuel `slapd.conf(5)`
- la page réplication du guide d'administration d'OpenLDAP ²³

8.2 Vérification

À la mise en place de la réplication, il est facile de s'assurer de son fonctionnement. D'abord, le répertoire hébergeant la base se remplit de nouveaux fichiers, ensuite, il suffit d'interroger le serveur secondaire pour s'assurer qu'il

23. <http://www.openldap.org/doc/admin24>

contient bien les entrées répliquées. Mais comment s'assurer que cette réplication persiste par la suite, et que les différents serveurs restent synchronisés ?

Nagios²⁴ est un outil de supervision bien connu. Sur le site Nagios Exchange²⁵, on trouve un greffon de surveillance de l'état de synchronisation de deux serveurs OpenLDAP, `check_syncrepl`. Il est écrit en python, et nécessite le paquetage `python-ldap`. Il est également disponible dans le paquetage `nagios-check_syncrepl` sous Mandriva. Une fois installé, il suffit de le lancer en lui donnant la base de recherche, et les URIs du fournisseur et du consommateur. Si les deux serveurs sont synchronisés, le résultat ressemble à ceci :

```
[guillaume@oberkamp ~]$ /usr/lib/nagios/plugins/check_syncrepl.py ldap://ldap1.domain.tld 1
2008-09-25 00:32:47,305 - check_syncrepl.py - INFO - Provider is: ldap1.domain.tld
2008-09-25 00:32:47,321 - check_syncrepl.py - INFO - Checking if consumer ldap2.domain.tld i
2008-09-25 00:32:47,327 - check_syncrepl.py - INFO - Provider and consumer exactly in SYNC
```

Si ce n'est pas le cas, par contre, le le résultat ressemble à cela :

```
[guillaume@oberkamp ~]$ /usr/lib/nagios/plugins/check_syncrepl.py ldap://ldap1.domain.tld 1
2008-09-25 00:32:47,305 - check_syncrepl.py - INFO - Provider is: ldap1.domain.tld
2008-09-25 00:32:47,305 - check_syncrepl.py - INFO - Provider is: ldap1.domain.tld
2008-09-25 00:32:47,321 - check_syncrepl.py - INFO - Checking if consumer ldap2.domain.tld i
2008-09-25 00:32:47,327 - check_syncrepl.py - INFO - Consumer NOT in SYNCH
2008-09-25 00:32:47,327 - check_syncrepl.py - INFO - Delta is 84 days, 0:50:21
```

En général, il suffit de relancer le consommateur, éventuellement après avoir vidé sa base, pour résoudre la plupart des problèmes.

Attention, pour la mise en oeuvre par nagios, il faut rediriger (option `-l`) dans un répertoire adéquat, ou mieux supprimer (option `-q`) l'enregistrement du résultat, et rajouter l'option `-n`. J'ai également rencontré des soucis avec des serveurs donnant une précision temporelle à la microseconde pour le marqueur de synchronisation, entraînant un crash. Un patch est disponible sur la page d'origine du greffon.

Le greffon `bb-openldap` pour `hobbit`, déjà présenté dans la recette 7.3, réalise également cette surveillance.

8.3 Redirection transparente des modifications

Un serveur esclave est par définition en lecture seulement, tous les changements se faisant exclusivement sur le maître. Or, dans certaines architectures, ce dernier n'est pas accessible aux utilisateurs. Comment faire dans ce cas pour leur permettre néanmoins d'effectuer les modifications les concernant, comme par exemple les changements de mot de passe ?

La solution présentée ici fait intervenir plusieurs fonctionnalités avancées de LDAP : les références LDAP (referrals), la délégation (proxy), et le greffon **chain**. Elle est donc assez complexe, mais parfaitement fonctionnelle.

24. <http://www.nagios.org>

25. <http://www.nagiosexchange.org/cgi-bin/page.cgi?g=Detailed\%2F2477.html>

Le début est trivial, puisqu'il suffit d'une simple directive **updateref** dans la configuration de l'esclave est suffisante :

```
updateref ldap://ldap1.domain.tld
```

Toute demande de modification se voit ainsi retournée une référence vers le serveur maître, comme par exemple ici :

```
[guillaume@oberkampf ~]$ ldappasswd -x -D uid=bar,ou=users,dc=domain,dc=tld -w password -s r
Result: Referral (10)
Referral: ldaps://ldap1.domain.tld
```

Certains clients LDAP peuvent être configurés pour suivre automatiquement cette référence, avec tous les problèmes de sécurité qui s'ensuivent. Mais de toute façon, le but poursuivi ici est de rendre la chose transparente pour l'utilisateur. Le greffon **chain** permet justement d'effectuer ce suivi automatiquement, au sein du serveur. Sur le serveur esclave, il faut donc rajouter la configuration suivante :

```
overlay          chain
chain-uri         "ldap://ldap1.domain.tld"
chain-idassert-bind bindmethod="simple"
                  binddn="cn=chain,ou=roles,dc=domain,dc=tld"
                  credentials="s3cr3t"
                  mode="self"
chain-idassert-authzFrom "*"
chain-tls         start
chain-return-error TRUE
```

La directive **chain-uri** met en place le suivi pour toutes les références correspondant à sa valeur. La directive **chain-idassert-bind** définit l'identifiant, le mot de passe et la méthode utilisée pour s'authentifier sur le serveur maître : en effet, le serveur esclave utilise un identifiant particulier, à la place de l'identifiant de la requête initiale. Si cette authentification réussie, par contre, c'est bien l'identifiant de cette requête initiale qui sera utilisée pour vérifier ses autorisations. Cette dissociation d'identité entre authentification et autorisation est rendue nécessaire par le fait que le mot de passe de l'utilisateur initial n'est plus disponible à cette étape, et qu'il n'est pas possible de rejouer son authentification sur le serveur maître. La directive **chain-idassert-authzFrom** autorise tous les identifiants à utiliser ce mandataire. Enfin, une connexion chiffrée est utilisée, et en cas d'échec, l'erreur est renvoyée au client à la place de la référence.

Enfin, il faut mettre en place le mandataire sur le maître. D'abord, il faut autoriser cette fonctionnalité, en définissant une politique de délégation :

```
# proxy authorization policy
authz-policy to
```

Cette directive, telle que définie ici, permet de s'authentifier auprès du serveur avec un identifiant donné, puis à utiliser un autre identifiant compatible

avec la valeur de l'attribut **authzTo** de cette entrée pour la détermination des autorisations. Nous avons besoin que l'identité **cn=chain,ou=roles,dc=domain,dc=tld**, utilisée par l'esclave, puisse ainsi prendre le rôle de n'importe quelle autre identité, il faut donc définir cette entrée de la façon suivante :

```
dn: cn=chain,ou=roles,dc=domain,dc=tld
objectClass: organizationalRole
objectClass: simpleSecurityObject
cn: chain
description: slave server proxy user
authzTo: dn:*
```

Une fois tout ceci en place, la tentative de modification précédente devrait réussir :

```
[guillaume@oberkampff ~]$ ldappasswd -x -D uid=bar,ou=users,dc=domain,dc=tld -w password -s r
[guillaume@oberkampff ~]$
```

Attention, l'attribut **authzTo** est un attribut opérationnel présent pour toute entrée. Autrement dit, un utilisateur capable de modifier son propre attribut est capable d'usurper l'identité de n'importe quel autre utilisateur... D'une manière générale, l'accès aux attributs de l'entrée d'un utilisateur par lui-même doit être strictement contrôlée, et n'autoriser la modifications que dans les cas pertinents :

```
# attributes
access to dn.subtree="ou=users,dc=domain,dc=tld" attrs=userPassword
    by self ssf=56 write
    by self peername.ip=127.0.0.1 write
    by anonymous ssf=56 auth
    by anonymous peername.ip=127.0.0.1 auth
    by * none

access to dn.subtree="ou=users,dc=domain,dc=tld"
    attrs=loginShell,gecos,displayName,description,telephoneNumber,mobile,mail,labeledURI,p
    by self write
    by * read

access to dn.subtree="ou=users,dc=domain,dc=tld"
    by * read
```

D'autres politiques de délégation, moins sensibles, peuvent également être adoptées. Plutôt qu'autoriser le mandataire à prendre n'importe quel identifiant, il peut être restreint à la branche utilisateur :

```
dn: cn=chain,ou=roles,dc=domain,dc=tld
objectClass: organizationalRole
objectClass: simpleSecurityObject
```

```
cn: chain
description: slave server proxy user
authzTo: dn.children:ou=users,dc=domain,dc=tld
```

Ou encore, la politique peut-être définie dans l'autre sens. Plutôt que de définir pour qui le mandataire peut se faire passer, on peut définir par qui un identifiant peut se faire représenter, en utilisant cette fois-ci l'attribut **authz-From**, et en changeant la configuration de slapd :

```
# proxy authorization policy
authz-policy from
```

Ces utilisateurs peuvent ainsi se faire représenter :

```
dn: uid=foo,ou=users,dc=domain,dc=tld
authzFrom: dn: cn=chain,ou=roles,dc=domain,dc=tld
```

```
dn: uid=bar,ou=users,dc=domain,dc=tld
authzFrom: dn: cn=chain,ou=roles,dc=domain,dc=tld
```

Cette politique offre un meilleur contrôle, mais nécessite plus de modifications dans les données.

Pour plus d'information, consulter :

- la page de manuel `slapd.conf`
- la page de manuel `slapo-chain(5)`
- la page de manuel `slapd-ldap(5)`
- la page greffon du guide d'administration d'OpenLDAP ²⁶
- la page stockage du guide d'administration d'OpenLDAP ²⁷

9 Kerberos

Le modèle d'authentification par défaut d'OpenLDAP est simple à mettre en oeuvre et à utiliser. Associé à une connexion chiffrée par SSL ou TLS, il permet également un niveau de sécurité satisfaisant. Mais il existe un autre modèle, plus robuste mais également plus complexe à mettre en oeuvre, basé sur SASL. Ce dernier est en fait un modèle générique, permettant d'utiliser différents mécanismes de façon transparente pour l'application. Parmi ceux-ci, on peut citer l'utilisation de certificats X.509, ou encore Kerberos. Ce dernier est un mécanisme d'authentification fort, qui nécessite la mise en place d'une infrastructure dédiée, et notamment un serveur de distributions de clés, le KDC (Key Distribution Center). Pour chaque application à laquelle il désire accéder, l'utilisateur obtient un ticket d'authentification auprès de ce KDC, et le présente à l'application. Si celle-ci arrive à le déchiffrer, elle a la garantie que l'utilisateur est bien celui qu'il prétend être.

Les recettes qui suivent montrent l'intégration d'un serveur OpenLDAP à cette infrastructure Kerberos, que l'on suppose déjà mise en place.

26. <http://www.openldap.org/doc/admin24>

27. <http://www.openldap.org/doc/admin24>

9.1 Authentification

La mise en oeuvre de Kerberos au sein d'OpenLDAP est relativement simple. Il suffit de créer un principal de la forme **ldap/host.domain.tld**, en utilisant le nom canonique du serveur LDAP, c'est-à-dire celui retourné par une interrogation DNS sur son adresse IP, et pas un quelconque alias DNS. En effet, le client effectue une normalisation du nom de l'hôte avant de demander un ticket au KDC. Puis il faut extraire la clé de ce principal dans le fichier keytab du serveur ldap, par exemple de la façon suivante :

```
[guillaume@oberkampf ~]$ kadmin -p admin/admin@DOMAIN.TLD ext_keytab ldap/host.domain.tld
```

Il faut ensuite s'assurer que le fichier keytab est bien lisible par le processus slapd, puis installer SASL et son greffon GSSAPI, et enfin relancer le serveur. Si tout est correct, le serveur devrait alors déclarer GSSAPI parmi les mécanismes d'authentification disponibles lorsque l'on interroge la racine DSE :

```
[guillaume@oberkampf ~]$ ldapsearch -x -b "" -s base supportedSASLMechanisms -LL
version: 1
```

```
dn:
supportedSASLMechanisms: GSSAPI
```

Il ne reste plus qu'à installer également SASL sur le client, à obtenir un ticket kerberos, puis à tester avec un client :

```
[guillaume@oberkampf ~]$ ldapsearch -Y GSSAPI -s base -LL
SASL/GSSAPI authentication started
SASL username: user@DOMAIN.TLD
SASL SSF: 56
SASL data security layer installed.
version: 1
```

```
dn: dc=domain,dc=tld
dc: domain
objectClass: top
objectClass: domain
```

Pour plus d'information, consulter :
– la page SASL du guide d'administration d'OpenLDAP ²⁸

9.2 Autorisation

Une fois Kerberos en place, l'utilisateur peut s'authentifier en présentant un ticket au serveur LDAP, à la place du couple identifiant/mot de passe. Néanmoins, il est alors affecté d'un identifiant distinct de son identifiant habituel, comme le montre la commande `ldapwhoami`, ce qui le prive des permissions attribuées à celui-ci :

²⁸. <http://www.openldap.org/doc/admin24>

```
[guillaume@oberkampf ~]$ ldapwhoami -Y GSSAPI
SASL/GSSAPI authentication started
SASL username: user@DOMAIN.TLD
SASL SSF: 56
SASL data security layer installed.
dn:uid=user,cn=gssapi,cn=auth
```

Pour effectuer la traduction de cet identifiant SASL en identifiant canonique, il faut utiliser la directive `authz-regexp` dans la configuration de `slapd` :

```
authz-regexp "uid=(^[^,]*) ,cn=gssapi,cn=auth" "uid=$1,ou=users,dc=domain,dc=tld"
```

Une fois ceci effectué, tout devrait rentrer dans l'ordre :

```
[guillaume@oberkampf ~]$ ldapwhoami -Y GSSAPI
SASL/GSSAPI authentication started
SASL username: user@DOMAIN.TLD
SASL SSF: 56
SASL data security layer installed.
dn:uid=user,ou=users,dc=domain,dc=tld
```

Pour plus d'information, consulter :
– la page de manuel `slapd.conf`

10 Conclusion

Cet article a présenté un certain nombre d'astuces utilisables avec OpenLDAP. Elles ont été en majorité mises en places et testées avec la branche 2.3. Le passage à la branche 2.4, qui est devenue la branche stable récemment, s'est fait sans problème majeur. Parmi les principales nouveautés de cette branche, il y a notamment la configuration dynamique, c'est à dire la possibilité de stocker la configuration dans l'annuaire lui-même plutôt que dans un fichier. Ceci devrait notablement faciliter la réalisation d'interfaces de configuration, puisqu'il est plus facile de parcourir et de modifier un annuaire normalisé que d'analyser un fichier possédant son propre format. Au passage, cela ouvre également des perspectives amusantes, comme par exemple de positionner des ACLs sur cette configuration. La documentation semble nettement s'améliorer également, avec une refonte du guide d'administration. Bref, l'outil progresse, et la transition se fait sans souci.

Même sans tenir compte de ces évolutions, OpenLDAP reste un sujet d'étude sans fin, dont il est difficile d'imaginer pouvoir maîtriser un jour l'intégralité des possibilités. Malgré l'expérience accumulée, je continue d'en apprendre tous les jours à son sujet. À titre d'exemple, j'avais soumis une première version de cet article à Denis en mai dernier. Mon courrier s'étant égaré au fond d'une boîte au lettre, c'est une nouvelle version qui voit le jour aujourd'hui. Mais dans l'intervalle, le volume de l'article a été multiplié par deux... D'autres sujets, que je n'ai pas eu l'occasion d'aborder jusqu'ici, restent encore à explorer, comme

par exemple les modules de stockage exotiques que sont slapd-meta, slapd-sql ou slapd-perl. De quoi écrire une troisième version un jour prochain ?

Pour finir, je tiens à remercier Buchan Milne et Andreas Hasenack, qui sont à l'origine d'une grande partie des informations présentées ici, ainsi que Mickaël Scherer pour sa relecture attentive.