slapd-access

Configuration d'accès pour slapd

Structure

access to <what> [by <who> [<access>] [<control>]]+

Donne accès à un jeu d'entrées et/ou attributs (what) par des demandeurs (who). Les ACL sont évaluées dans l'ordre auxquelles elles apparaissent. lors qu'un what match, who est vérifié. puis les clauses access et control sont évaluées

Le champ WHAT

Spécifie l'entité à laquelle cette ACL s'applique. Il peut avoir les formes suivante :

```
dn[.<dnstyle>]=<dnpattern>
filter=<ldapfilter>
attrs=<attrlist>[ val[/matchingRule][.<attrstyle>]=<attrval>]
```

avec :

```
<dnstyle>={{exact|base(object)}|regex|one(level)|sub(tree)|children}
<attrlist>={<attr>|[{!|@}]<objectClass>}[,<attrlist>]
<attrstyle>={{exact|base(object)}|regex|one(level)|sub(tree)|children}
```

dn=<dnpattern> Sélectionne les entrées basées sur leur contexte de nommage

<dnstyle> (optionnel). **Base** (synonymes de baseObject) ou exact (alias de base) indique l'entrée dont le DN est égal à <dnpattern>. **one** (synonyme de onelevel) indique toutes les entrées immédiatement sous le <dnpattern>, **sub** (synonyme de subtree) indique toutes les entrées dans la sous-arborescence de <dnpattern>, **children** indique toutes les entrées subordonnées à <dnpattern> Si dnstyle est un regex, <dnpattern> est une expression étendue POSIX et match une représentation du DN.

filter=<ldapfilter> Sélectionne les entrées basée sur un filtre LDAP

attrs=<attrlist> sélectionne les attributs auxquels la règle s'applique. Le signe **+** indique l'accès à l'entrée elle-même. **children** indique l'accès à l'entrée de l'enfant. Les classes d'objet peuvent être spécifiés également. Les noms préfixés par '**@**' sont directement traités comme nom de classe d'objet. un nom préfixé par '**!**' et traité comme classe d'objet mais la règle affecte les attributs non requis ni permis. sans cette liste, assume **attrs=@extensibleObject**

attrs=<attrlist> [val [/matchingRule] [.<attrstyle>] =<attrval>] Cette forme spécifie un accès à une valeur particulière d'un simple attribut. seul l'attribut est donnée.

dn, filter, attrs sont additifs. les sous-match résultant de regex peuvent être dé-référencés dans **<who>** avec la syntaxe ${v_n_} où **_n_** est le numéro de sous-match.

Le champ WHO

Indique à qui la règle s'applique. Plusieurs who peuvent apparaîtrent dans l'acl. Il a la forme :

*

```
anonymous
users
self[.<selfstyle>]

dn[.<dnstyle>[,<modifier>]]=<DN>
dnattr=<attrname>

realanonymous
realusers
realself[.<selfstyle>]

realdn[.<dnstyle>[,<modifier>]]=<DN>
realdnattr=<attrname>

group[/<objectclass>[/<attrname>]][.<groupstyle>]=<group>
peername[.<peernamestyle>]=<peername>
sockname[.<style>]=<sockname>
domain[.<domainstyle>[,<modifier>]]=<domain>
sockurl[.<style>]=<sockurl>
set[.<setstyle>]=<pattern>

ssf=<n>
transport_ssf=<n>
tls_ssf=<n>
sasl_ssf=<n>

dynacl/<name>[/<options>][.<dynstyle>][=<pattern>]
avec:
<style>={exact|regex|expand}
<selfstyle>={level{<n>}}
<dnstyle>={{exact|base(object)}|regex|one(level)|sub(tree)|children|level{<n>}}
<groupstyle>={exact|expand}
<peernamestyle>={<style>|ip|ipv6|path}
<domainstyle>={exact|regex|sub(tree)}
<setstyle>={exact|expand}
<modifier>={expand}
<name>=aci <pattern>=<attrname>]
```

***** Indique tout le monde

realanonymous Les mots clé préfixés par real agissent comme leur homologue non préfixés. La vérification se produit avec le DN authentification et le DN authorisation

anonymous référence les clients non authentifiés

users Références les clients authentifiés

self Référence l'entrée elle-même (l'entrée demandée et l'entrée qui requête doit correspondre). Accepte le style level{<n>}, où _n_ indique que l'ancêtre du DN est utilisé dans le match. Une valeur positive indique que le <n>-ième ancêtre du DN de l'utilisateur est considéré ; une valeur négative indique que le <n>-ième ancêtre de la cible doit être considéré. (ex : "by self.level{1} ..." matche quand l'objet "dc=example,dc=com" est accédé par cn=User,dc=example,dc=com") (ex : "by self.level{-1} ..." matche quand le même user accède à "ou=Address Book,cn=User,dc=example,dc=com"

dn=<DN> L'accès est donnée au DN qui matche.

dnstyle (optionnel) autorise le même choix que pour WHAT. le style regex exploit la substitution de sous-match dans le dn.regex de WHAT en utilisant la forme $<digit>, de 0 à 9 où 0 match toute la chaîne. ${<digit>+} pour les sous-matches > 9, ${v<digit>+} pour la substitution de chaîne de valeur d'attributs. le $ de fin de chaîne doit être spécifié par '$$'

access to dn.regex="^(.+,)?uid=([^,]+),dc=[^,]+,dc=com$" by dn.regex="^uid=$2,dc=[^,]+,dc=com$$" write
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$" by dn.exact,expand="uid=$2,dc=example,dc=com" write
access to dn.regex="^(.+,)?uid=([^,]+),dc=([^,]+),dc=com$" by dn.exact,expand="uid=$2,dc=$3,dc=com" write

level{n} est une extension de la forme onelevel, où l'ancêtre n est le pattern. Donc level{1} est équivalent à onelevel et level{0} vaut base

dnattr=<attrname> L'accès est donné aux requêtes dont le DN est listé dans l'entrée accédée sous l'attribut <attrname>

group=<group> L'accès est donné aux requêtes dont le DN est listé dans l'entrée group dont le DN est donné par <group> <objectclass> et <attrname> donnent la classe objet et l'attribut des membres.(défaut : groupOfNames et member). <style> peut être expand qui signifie que <group> sera étendu comme remplacement (mais non comme expression régulière), et exact, qui signifie que le match exacte sera utilisé. Pour des groupes statiques, le type d'attribut spécifié doit avoir la syntaxe DN ou NameAndOptionalUID. pour les groupes dynamiques, le type d'attribut doit être un sous-type de labeledURI

peername=<peername> L'IP de l'hôte contactant (sous la forme IP=<ip> :<port> ou IP=[<ipv6>] :<port>) ou le nom de l'hôte contactant (PATH=<path>)

sockname=<sockname> Le nom du fichier de pipe nommé

domain=<domain> Le nom d'hôte du contactant

sockurl=<sockurl> l'URL du contactant, sont comparés avec le pattern pour déterminer l'accès. Le style décrit pour group et regex s'appliquent. Le cas spécial pour ip : <peername>=<ip>[%<mask>][{<n>}] où n est le nom du port optionnel (ex : peername.ip=192.168.1.16%255.255.255.240{9009}). (ex : domain.subtree=example.com match www.example.com)

set=<pattern>

dynacl/<name>[/<options>] [.<dynstyle>] [=<pattern>] Indique que la vérification de l'accès est déléguée à une méthode définies indiquée par <name>, qui peut être enregistrée en temps réel au moyen de déclarations moduleload. <options>, <dynstyle> et <pattern> sont optionnels et sont passés directement à la routine de parsing enregistrée

dynacl/aci [=<attrname>] Signifie que le contrôle d'accès est déterminé par les valeurs dans attrnames de l'entrée elle-même. <attrname> indique quel type d'attribut maintient l'ACI dans l'entrée. Par défaut, OpenLDAPaci est utilisé

ssf=<n>

transport_ssf=<n>

tls_ssf=<n>

sasl_ssf=<n> Définissent le SSF minimum pour obtenir l'accès

Le champ ACCESS

Optionnel. Détermine le niveau d'accès ou le privilège que who aura. Il a la forme :

```
<access> ::= [[real]self]{<level>|<priv>}
où:
<level> ::= none|disclose|auth|compare|search|read|{write|add|delete}|manage
<priv> ::= {=|+|-}{0|d|x|c|s|r|{w|a|z}|m}+
```

self Permet des opération spéciales pour le DN autorisé

realself Réfère au DN authentifié

level ce modèle d'accès s'assure d'une interprétation incrémentale des privilèges. Les niveaux possibles sont none (aucun accès, disclose (divulgation d'information en cas d'erreur), auth (permettre l'authentification), compare, search, read, write (add+delete) et manage (inclus des accès administratifs).

priv le modèle d'accès priv se base sur des paramètres explicites de privilèges pour chaque clause. = réinitialise les accès définis précédemment, + et - ajoute/supprime des privilèges. m (manage), w (write), a (add), z (delete), r (read), s (search), c (compare), x (authentification), d (disclose), 0 (aucun privilège) (défaut : +0)

Le champ Control

Optionnel. Contrôle le flux de règles d'accès. Peut être :

stop Stop en cas de match

continue Permet à d'autres clause <who> dans le même <access> d'être considérés

break Permet à d'autres clauses <access> qui matchent la même cible d'être traités

exemple break :
access to dn.subtree="dc=example,dc=com" attrs=cn by * =cs break
access to dn.subtree="ou=People,dc=example,dc=com" by * +r
exemple continue :
access to dn.subtree="dc=example,dc=com" attrs=cn by * =cs continue by users +r

Scopes

base correspond seulement à l'entrée avec le DN fournit

one correspond aux entrées dont le parent est le DN fournit

subtree correspond à toutes les entrées dans l'arbre dont le root est le DN fournit

children correspond à toutes les entrées sous le DN (mais pas l'entrée nommée par le DN)

Par exemple, si l'annuaire contient les entrées nommées :

```
o=suffix
cn=Manager,o=suffix
ou=people,o=suffix
uid=kdz,ou=people,o=suffix
cn=addresses,uid=kdz,ou=people,o=suffix
uid=hyc,ou=people,o=suffix
```

alors :

dn.base="ou=people,o=suffix" match 2 ;

dn.one="ou=people,o=suffix match 3, and 5 ;

dn.subtree="ou=people,o=suffix match 2, 3, 4, and 5 ; and

dn.children="ou=people,o=suffix match 3, 4, and 5.

cibles

```
*_____________________________All, including anonymous and authenticated users
anonymous____________________Anonymous (non-authenticated) users
users________________________Authenticated users
self_________________________User associated with target entry
dn[.<basic-style>]=<regex>_Users matching a regular expression
dn.<scope-style>=<DN>______Users within scope of a DN
```

Correspondance des droits

```
Level_____Privileges______Description
none______0_______________no access
disclose__d_______________needed for information disclosure on error
auth______dx______________needed to authenticate (bind)
compare___cdx_____________needed to compare
search____scdx____________needed to apply search filters
read______rscdx___________needed to read search results
```

```
write_____wrscdx__________needed to modify/rename
manage____mwrscdx_________needed to manage
```

Opérations requises

Les opérations nécessitent différents privilèges sur différentes portions d'entrées.

add nécessite le privilège add sur le pseudo-attribut de l'entrée à ajouter, et add sur le pseudo-attribut children du parent de l'entrée.

bind quand les accréditifs sont stockés dans l'annuaire, nécessite les privilèges auth sur l'attribut où sont stockés ces accréditifs.

Compare nécessite compare sur l'attribut à comparer

delete nécessite delete sur le pseudo-attribut de l'entrée à supprimer et sur le pseudo-attribut children du parent de l'entrée

modify nécessite write (add pour ajouter, delete pour supprimer, les 2 pour modifier)

modrdn nécessite write sur le pseudo-attribut de l'entrée à supprimer, delete sur le pseudo-attribut Children de l'ancien parent de l'entrée, et add sur le pseudo-attribut Children du nouveau parent de l'entrée. delete est aussi requis sur les attributs qui sont présent dans l'ancien RDN si deleteoldrdn et à 1.

search nécessite search sur le pseudo-attribut entry du searchBase. Les entrées résultantes sont ensuite testé sur read sur le pseudo-attribut entry et sur chaque valeurs de chaque attribut demandé pour chaque referral utilisé pour générer des références continues, read sur le pseudo-attribut entry et les attributs du referral.

authzID, proxyAuthz necessitent auth sur tous les attributs présents dans la recherche et sur authzTo et/ou authzFrom de l'identité autorisant.

Exemples

Pour matcher la sous-arborescence désirée, la règle serait :
access to dn.regex="^(.+,)?dc=example,dc=com$" by ...
Pour des raisons de performance, il est mieux d'écrire :
access to dn.subtree="dc=example,dc=com" by ...
En écrivant des règles submatch, il peut être préférable d'éviter l'utilisation de <dnstyle> :
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
by dn.regex="^uid=$2,dc=example,dc=com$$" write
by ...
Cependant, c'est plus efficace :
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
by dn.exact,expand="uid=$2,dc=example,dc=com" write
by ...