draft-ietf-ldapext-acl-model-08

Modèle de contrôle d'accès

La capacité d'accéder aux informations de manière sûre via le réseau est nécessaire pour un déploiement réussi. Actuellement, LDAP ne définis pas de modèle de contrôle d'accès, mais il est nécessaire pour assurer un accès sécurisé conforme, réplication et gestion via des implémentations LDAP hétérogènes. L'objectif majeur est de fournir un modèle de contrôle d'accès simple, exploitable, implémentable et sécurisé pour l'accès au contenu des annuaires LDAP tout en fournissant la flexibilité appropriée pour les besoins de l'Internet et des environnements d'entreprise. Ce document définis le modèle, le schéma, et le contrôle LDAP.

Ce modèle ne (et ne peut pas) spécifie pas pleinement le fonctionnement du modèle dans un environnement distribué (ex : propager les informations de contrôle d'accès entre les serveurs).

Les mécanismes de contrôle d'accès évaluent les demandes pour accéder aux ressources protégées et prend des décisions d'autoriser ou non l'accès. Pour faire ces décisions, pour une demande, un mécanisme de contrôle d'accès doit évaluer des données de stratégie. Cette stratégie décris des caractéristiques de sécurité requises par le sujet et des règles qui gouvernent l'utilisation le l'objet cible.

Aucun mécanisme n'est définis dans ce document pour stocker les informations de contrôle d'accès. Le modèle de contrôle d'accès définis :

- Les flux de protocole LDAP existant pour les opérations ldap sont utilisés pour manipuler les informations de contrôle d'accès. Ces même flux s'appliquent quand des ACI sont transmis durant la réplication. Un jeu de permission et leur sémantique est définis. Il y a une contrôle définis, GetEffectiveRights que les clients utilisent pour gérer et administrer les contrôles d'accès des serveurs.
- Les attributs et classes pour la portabilité des informations de contrôle d'accès aux applications. Les applications ldap portables devraient seulement utiliser ce modèe d'ACI. 2 attributs de contrôle d'accès (entryACI et subtreeACI) sont utilisé comme entrée à l'API ldap pour que les informations de contrôle d'accès puisse être adressés uniformément et indépendamment de la manière dont ces informations sont accédés et stockés dans le serveur. Une classe subentry (ldapACISubEntry) et un jeu d'attributs (supportedAccessControlSchemes, accessControlSchemes) sont utilisés pour identifier les mécanismes de contrôle d'accès supportés par un serveur.
- Un mécanisme pour contrôler l'accès aux ACI : les attributs opérationnel entryACI et subtreeACI sont utilisés pour contrôler les accès aux ACI.

Les serveurs peuvent supporter plusieurs mécanismes de contrôle d'accès, mais doivent être capable de supporter le mécanisme LDAP dans le DIT scopé par les rootDSE (tout de DIT du serveur) et devrait être capable de supporter le mécanisme ldap dans une portion arbitraire (subtree) de ce DIT.

L'attribut accessControlSchemes dans le ldapACISubEntry indique quels mécanisme de contrôle d'accès sont effectifs pour le scope de ce subentry. L'attribut supportedAccessControlSchemes dans le rootDSE indique quels mécanismes de contrôle d'accès sont supportés par le serveur. Ces mécanismes sont effectifs dans le DIT du serveur sauf s'il sont remplacés par un mécanisme définis dans un ldapACISubEntry quelque part dans le DIT.

Changer les valeurs de supportedAccessControlSchemes ou accessControlSchemes change les mécanismes actifs pour le scope de ces attributs. Via l'utilisation de supportedAccessControlSchemes dans le rootDSE et accessControlSchemes dans ldapACISubEntry, il est possible de lancer plusieurs mécanismes dans le même subtree ou des subtree séparés.

Attributs de mécanisme de contrôle d'accès

2 attributs sont définis pour identifier quels mécanismes sont supportés par un serveur donné et par un subtree donné : **supportedAccessControlSchemes** et **accessControlSchemes**.

Attribut RootDSE pour les mécanismes

Le serveur utilise supportedAccessControlSchemes dans le rootDSE pour difuser les mécanismes supportés. Cet attribut est une liste d'OID, chacun identifiant un mécanisme de contrôle d'accès supporté par le serveur. Par défaut, il y a également les mécanismes actifs dans les subtrees.

```
(<OID to be assigned>
 NAME 'supportedAccessControlSchemes'
 DESC list of access control mechanisms supported by this directory server
 EQUALITY objectIdentifierMatch
 SYNTAX OID
 USAGE dSAOperation
)
```

Mécanisme de contrôle d'accès subentry

Un contexte de nommage doit fournir des informations sur les mécanismes de contrôle d'accès qui sont actifs pour la portion de l'espace de nom. Cette information est contenue dans un subentry (ldapACISubEntry). Il peut être utilisé pour définir le scope d'un mécanisme de contrôle d'accès. Les valeurs maintenus dans le rootDSE, sont les mécanismes actifs dans les subtrees sous le root à moins d'être substitué dans un ldapACISubEntry plus bas dans l'arborescence maintenue par le serveur. Le scope de cet ldapACISubEntry est à la fin du subtree maintenu par ce serveur ou jusqu'à ce qu'un autre ldapACISubEntry soit rencontré. la classe ldapACISubEntry est définis :

```
( <OID to be assigned>
 NAME 'ldapACISubEntry'
 DESC 'LDAP ACI Subentry class'
 SUP ldapSubEntry STRUCTURAL
 MUST ( accessControlSchemes )
)
```

accessControlSchemes doit être dans chaque ldapACISubEntry associé avec un contexte de nommage dont le mécanisme de contrôle d'accès est différent des contextes de nommage adjacents supportés par ce serveur. **accessControlSchemes** liste les valeurs qui définissent les mécanismes de contrôle d'accès effectifs pour le scope de ce subentry. Bien qu'en général cet attribut définis un seul mécanisme, il peut en contenir plusieurs.

```
(<OID to be assigned>
 NAME 'accessControlSchemes'
 DESC list of access control mechanisms supported in this subtree
 EQUALITY objectIdentifierMatch
 SYNTAX OID
 USAGE dSAOperation
)
```

Définitions

La syntaxe et attributs pour les informations de contrôle d'accès, entryACI et subtreeACI, sont définis :

```
(<OID-aci-syntax>
 DESC 'attribute syntax for LDAP access control information'
)

(<OID to be assigned>
```

```
 NAME 'entryACI'
 DESC 'ldap access control information, scope of entry'
 EQUALITY directoryComponentsMatch
 SYNTAX <OID-aci-syntax>
 USAGE directoryOperation
)

(<OID to be assigned>
 NAME 'subtreeACI'
 DESC 'ldap access control information, scope of subtree'
 EQUALITY directoryComponentsMatch
 SYNTAX <OID-aci-syntax>
 USAGE directoryOperation
)
```

Le but de ces définitions d'attribut est de définir un format commun et d'avoir une séparation des responsabilités pour permettre à différentes personnes d'administrer des différents types d'attributs. Tout serveur LDAP devrait être capable de traduire l'attribut définis en requêtes significatives. Chaque serveur devrait être capable de comprendre l'attribut ; Il ne devrait pas y avoir d'ambiguïté dans aucune partie de la syntaxe.

Alors que le but final est d'avoir un modèle commun entre différentes implémentations de serveur LDAP, les définitions d'attribut seul n'auront pas forcement un traitement d'ACL identique entre les serveurs. Les règles d'applicabilité et de précédence pour prendre les décisions d'accès sont définis plus bas dans cette section. Les sémantiques sur la manière dont un serveur interprète da syntaxe ACI sont définis plus bas dans ce document. Additionnellement, tandis que le serveur doit reconnaître et agir sur l'attribut quand il reçoit de flux, il n'y a aucun requis pour le serveur pour stocker physiquement ces attributs sous cette forme.

Les définitions d'attribut maintiennent une hypothèse selon laquelle le serveur de réception supporte l'héritage d'ACI dans le modèle de sécurité. Si le serveur ne supporte pas l'héritage, le serveur de réception doit étendre toute information d'héritage basé dur le flag de scope.

Les attributs sont définis pour que les ACI puissent être adressés dans un serveur d'une manière indépendante de l'implémentation. Ces attributs sont utilisés dans les API LDAP, dans la sortie LDIF de l'ACI et en transfert d'ACI durant la réplication. Ces attributs peuvent être requêtés ou définis dans tous les objets de l'annuaire. Les définitions et notation ABNF sont donnés ci-dessous.

Représentation de chaîne UTF-8

L'encodage des chaînes LDAP des valeur de la syntaxe ACI est décrite :

```
ACI = rights "#" attr "#" generalSubject

rights = (("grant:" / "deny:") permissions) /
 ("grant:" permissions ";deny:" permissions)

permissions = 1*(permission)
permission = "a" / ; add
 "d" / ; delete
 "e" / ; export
 "i" / ; import
 "n" / ; renameDN
 "b" / ; browseDN
 "v" / ; view (entry level read permission)
 "t" / ; returnDN
 "r" / ; read
 "s" / ; search filter
 "p" / ; search filter (presence only)
 "w" / ; write (mod-add)
```

```
 "o" / ; obliterate (mod-del)
 "c" / ; compare
 "m" / ; make
 "u" / ; unveil (disclose on error permission)
 "g" ; getEffectiveRights

; les permissions r, w, o, s, p, c, m fonctionnent sur les attributes
; les permissions a, d, e, i, n, b, v, t, u, g fonctionnent sur les entrées
; les permissions pour les attributs et permissions pour les entrées ne sont jamais trouvés dans un simple
ACI

attr = "[all]" / "[entry]" / (attribute *("," attribute))

attribute = AttributeDescription

generalSubject = context pureSubject

context = "authnLevel:" authnLevel ":"

pureSubject = anySubject / machineSubject / idBasedSubject

anySubject = "public:"

machineSubject = "ipAddress:" ipAddressRange *( "," ipAddressRange ) /
 "dns:" partialdomainname *( "," partialdomainname )

partialdomainname = [ "*." ] domainname

idBasedSubject = thisSubject /
 oneSubject /
 setOfSubjects

thisSubject = "this:"
oneSubject = ( "authzId-" authzId )
setOfSubjects = ( "role:" dn ) /
 ( "group:" dn ) /
 ( "subtree:" dn )

authnLevel = "none" /
 "weak" /
 "limited" /
 "strong"

authzId = dnAuthzId / uAuthzId

; distinguished-name-based authz id.
dnAuthzId = "dn:" dn

dn = utf8string

; unspecified userid, UTF-8 encoded.
uAuthzId = "u:" userid
userid = utf8string ; syntax unspecified
; A utf8string is defined to be the UTF-8 encoding of
; one or more ISO 10646 characters.

ipAddress = IPv6address

; following is excerpted from [IPV6]
```

```
IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT

hexpart = hexseq | hexseq "::" [ hexseq ] | "::" [ hexseq ]
hexseq = hex4 *( ":" hex4)
hex4 = 1*4HEXDIG

ipAddressRange = ipAddress [ HYPHEN ipAddress ] ; IPv6 address range

domainname = domaincomponent *( "." domaincomponent )

OUTER = ALPHA / DIGIT
INNER = ALPHA / DIGIT / HYPHEN
HYPHEN = %x2D
domaincomponent = OUTER [ *61( INNER ) OUTER ]
```

Noter que les virgules suivant "public" et "this" existent seulement pour simplifier le parsing des chaînes. Si on tente d'ajouter un ACI avec un authz qui n'est pas supporté, le serveur doit rejeter cette valeur entryACI ou subEntryACI. Voir la section "Examples d'ACI" pour les examples de représentation de chaîne.

Représentation binaire

```
LDAP-Access-Control-Model
DEFINITIONS EXTENSIBILITY IMPLIED ::=
BEGIN

IMPORTS
 AttributeType, DistinguishedName, CONTEXT
  FROM InformationFramework; - from [X501]

ACI ::= SEQUENCE {
 rights SEQUENCE {
  grant Permissions OPTIONAL,
  deny [1] Permissions OPTIONAL }
   (WITH COMPONENTS { ..., grant PRESENT } |
   WITH COMPONENTS { ..., deny PRESENT }),
   - at least one of grant or deny must be present -
 attr CHOICE {
  all NULL,
  entry [1] NULL,
  attributes SET (1..MAX) OF AttributeTypeAndOptions },
 authnLevel ::= ENUMERATED {
  none (0),
  weak (1),
  limited (2),
  strong (3)}
 subject GeneralSubject
}


-Une représentation X.500 pour une description d'attribut LDAP
AttributeTypeAndOptions ::= SEQUENCE {
 type AttributeType,
 type-name UTF8String OPTIONAL,
options SEQUENCE SIZE (1..MAX) OF CONTEXT.&Assertion OPTIONAL
```

```
}
GeneralSubject ::= CHOICE {
 anySubject NULL,
 machineSubject [1] MachineSubject,
 idBasedSubject [2] IDBasedSubject
}

MachineSubject ::= CHOICE {
 ipAddress IPAddress,
 dns [1] DomainName
}

IPAddress ::= UTF8String

DomainName ::= UTF8String

IDBasedSubject ::= CHOICE {
 thisSubject NULL,
 oneSubject [1] OneSubject,
 setOfSubjects [2] SetOfSubjects
}

OneSubject ::= CHOICE {
 dn DistinguishedName,
 user UTF8String
}

SetOfSubjects ::= CHOICE {
 role DistinguishedName,
 group [1] DistinguishedName,
 subtree [2] DistinguishedName
}


Permissions ::= BIT STRING {
 add (0),
 delete (1),
 export (2),
 import (3),
renameDN (4),
 browseDN (5),
 viewEntry (6),
 returnDN (7),
 read (8),
 search (9),
 searchPresence (10),
 write (11),
 obliterate (12),
 compare (13),
 make (14),
 unveil (15),
 getEffectiveRights (16)
 (CONSTRAINED BY { - at least one bit must be set - })
}
```

Les permissions read, write, obliterate, search, searchPresence, compare, make fonctionnent sur les permissions d'attributs. add, delete, export, import, renameDN, browseDN, viewEntry, returnDN, unveil, getEffectiveRights fonctionnent sur les entrées.

Les composants des attributs entryACI et subtreeACI

Cette section définis les composants qui comprennent les attributs d'informations de contrôle d'accès, entryACI et subtreeACI. Les information dans entryACI s'appliquent seulement à l'entrée dans laquelle elle est contenue. Les informations dans subtreeACI s'appliquent à chaque entrée sous le subtree sauf si un entryACI spécifique ou un subtreeACI est rencontré. L'utilisation des ACI prescritives et le scoping via l'utilisation d'un ldapACISubEntry n'est pas décris dans ce document.

Droits d'accès et permissions

Les droits d'accès peuvent être appliqués à tout un object ou à des attributs de l'objet. L'accès peut être donné ou refusé. Une ou les deux actions "grant" | "deny" peuvent être utilisé en créant ou mettant à jour un entryACI et subtreeACI.

Permissions que s'appliquent aux attributs :

r Read - Lire les valeurs de l'attribut

w Write - Modifier-Ajouter des valeurs

o Obliterate - Modifier-Suprimer des valeurs

s Search - Rechercher des entrées avec les attributs spécifiés

p SearchPresence - Filtres de présence uniquement

c Compare - Comparer des attributs

m Make - Créer des attributs sous une nouvelle entrée sous cette entrée

Permissions que s'appliquent aux entrées :

a Add - Ajouter une entrée sous cette entrée

d Delete - Supprimer cette entrée

e Export - Exporter l'entrée et ses subordonnées à un nouvel emplacement

i Import - Importer une entrée et ses subordonnées sous cette entrée depuis un autre emplacement

n RenameDN - Renommer un DN de l'entrée

b BrowseDN - Parcourir un DN de l'entrée

v ViewEntry - Une permission de lire au niveau de l'entrée

t ReturnDN - Permet au DN de l'entrée d'être inclus dans le résultat de l'opération.

u Unveil - Permission discloseOnError

g GetEffectiveRights - permet de lire les Permissions effectives

Attributs

Un attribut décrit un nom d'attribut sous la forme d'un OID pour cet <attr>. Si l'OID réfère à un attribut non définis dans le schéma du serveur, le serveur devrait reporter une erreur. L'utilisation de "[entry]" ou "[all]" aide à focaliser sur les permissions de l'entrée ou de l'attribut facilement.

"[entry]" signifie que les permissions s'appliquent à tout l'objet. Peut être utilisé par exemple pour supprimer ou ajouter un objet. Quand utilisé dans un subtreeACI, les permissions spécifiées s'appliquent à chaque entrée dans le subtree.

"[all]" signifie que le jeu de permissions d'applique à tous les attributs de l'entrée. Quand utilisé dans un subtreeACI, "[all]" s'applique à tous les attributs de chaque entrée dans le subtree. Si le mot clé "[all]" et un autre attribut sont spécifiés dans une ACI, le jeu de permission de plus spécifique pour l'attribut écrase le jeu de permission le moins spécifique pour "[all]".

Sujets et authentification associé

Les sujets suivants sont définis et doivent être supportés :

- authzId
- group, définis comme DN d'une entrée groupOfNames ou groupOfUniqueNames
- role, affirme la position de l'organisation d'un sujet et le droit d'un sujet à effectuer les opérations nécessaires à cette position dans l'organisation. Cette implémentation définis comment l'association entrée un authorization id et le dn du role est faite
- subtree, définis comme DN d'un nœud non-terminal dans le DIT
- ipAddress, au format texte IPv6
- dnsName, un nom de domaine ou un nom de domaine wildcard
- public, définis comme accès publique
- this, définis comme l'utilisateur dont le nom match l'entrée accédé

D'autres parties peuvent définir d'autres sujets. Il est de la responsabilité de ces parties de fournir la définition. Ajouter de nouveaux sujets peut inhiber d'interopérabilité.

Un sujet peut être qualifié par le niveau d'authentification requis pour accéder à un attribut ou une entrée donnée. authnLevel définis le niveau d'authentification minimum du demandeur requis pour un ACI donné. Les niveaux d'authentification définis, dans l'ordre croissants d'authentification sont :

none Aucun nom et aucun mot de passe, ou un nom mais pas de mot de passe.

weak Les mécanismes d'authentification qui sont considéré vulnérables aux attaques passives et actives. Ex : l'authentification simple.

limited Les mécanismes d'authentification qui sont protégés contre les attaques passives mais pas les attaques actives. Ex : DIGEST-MD5

strong Les mécanismes d'authentification qui sont protégés contre les attaques passives et actives. Ex : Kerberos.

Le authnLevel s'applique à un sujet comme suit :

- Si l'ACI est donné, authnLevel s'applique si le sujet est authentifié à ce niveau ou plus
- Si l'ACI est refusé, authnLevel s'applique à ce sujet s'il est authentifié à ce niveau et à tous les autres sujets authentifiés avec les niveaux inférieurs.

Le niveau d'authentification est toujours spécifié dans un ACI. Pour grant, celà signifie que l'accès est donné si vous prouvez votre authentification via un mécanisme d'authentification fort, tel que la signature numérique. Pour deny, cela signifie que l'accès est refusé si vous êtes Mr. X et que vous le prouvez avec une signature numérique. Si vous n'êtes pas Mr. X votre accès n'est pas refusé seulement si vous pouvez le prouver avec une signature numérique. En d'autres termes, toute personne qui s'authentifie avec une signature numérique gagne l'accès excepté Mr. X.

Une catégorisation recommandée des mécanismes d'authentification par niveau peut être offert séparément. Pour chaque mécanisme catégorisé dans les recommandations, les implémentations ne devraient pas assigner un niveau d'authentification supérieur, mais peuvent assigner un niveau d'authentification inférieurs. Pour les mécanismes non couverts par les recommandations, l'implémentation devrait spécifier un niveau conservateur. Les implémentations devraient fournir un moyen pour les administrateurs de désactiver et/ou abaisser le niveau d'authentification associé avec un mécanisme.

2 sujets intéressant non explicitement inclus, mais peuvent être composés en utilisant le sujet et authnLevel associé avec un mécanisme.

anonymous authnLevel=none + anySubject(public)

authenticated authnLevel=weak + anySubject(public)

Décision de contrôle d'accès

Le but ultime du modèle de contrôle d'accès est de définir la manière dans laquel un serveur LDAP détermine une décision de contrôle d'accès pour une opération LDAP donnée. En fait, un demandeur peut nécessiter de nombreuses permissions individuelles pour être

autorisé à effectuer l'opération LDAP. Dans cette section on introduit les concepts et algorithmes qui permettent au serveur de décider si un demandeur à une permission individuelle sur un ressource individuelle. Les concepts nécessaires sont premièrement, les paramètres qui doivent être fournis dans l'ordre pour l'algorithme de décision de contrôle d'accès. pour déterminer si l'accès est permis ou non.

Deuxièmement, on définis précisément quand un ACI donné sera impliqué dans une décision de contrôle d'accès. Troisièmement, ce modèle définis certaines règles de précédence que l'algorithme utilisant lorsqu'il a plus d'un ACI. Finalement, on peut définir l'algorithme de décision de contrôle d'accès qui va déterminer la décision d'accès.

Les paramètres de l'algorithme

L'algorithme de décision répond à la question "est-ce que le demandeur a la permission spécifiée sur la ressource spécifée ?". La ressource peut être une entrée ou un attribut dans une entrée, donc on caractérise la ressource comme ceci : **targetEntry, targetAttribute OPTIONAL**.

Le demandeur est identifié principalement par son authorization ID (qui peut être omis si le demandeur est anonyme), mais inclus également les informations de contexte sur le demandeur pour qu'il soit représenté comme ceci : **authzId OPTIONAL, ipAddress, dnsName, authnLevel**.

La permission est une des permissions définis par le modèle. Donc les paramètres de l'algorithme de décision de contrôle d'accès, que nous appelons les paramètres de décision sont :

ressource (targetEntry, targetAttribute OPTIONAL)

permission permissionParameter

requestorSubject (authzId OPTIONAL, ipAddress, dnsName, authnLevel)

Si **permissionParameter** est un paramètre de niveau attribut, alors targetAttribute doit être spécifié ; s'il c'est une permission de niveau entrée, targetAttribute est ignoré.

La sortie est soit "Access Allowed" soit "Access Denied".

Règles d'applicabilité

Les règles d'applicabilité définissent si une valeur d'ACI donné, ou des portions, s'applique à certains paramètres de décision.

Règles d'applicabilité pour le type de scope

Ces règles détermine si un ACI spécifique d'applique à la partie targetEntry des paramètres de décision.

- Si l'ACI en question est un entryACI, alors l'ACI s'applique à la ressource si l'ACI est un attribut de l'entrée targetEntry.
- Si l'ACI en question est un subtreeACI, alors l'ACI s'applique à la ressource si targetEntry fait partie du subtree définis par l'entrée contenant l'ACI.

Règles d'applicabilité pour les permissions

- Si permissionParameter est une permission au niveau entrée, alors l'ACI en question s'applique si permissionParameter est mentionné dans la liste des permissions de l'ACI.

- Si permissionParameter est une permission au niveau attribut, alors l'ACI en question s'applique si permissionParameter est mentionné dans la liste des permissions et la liste des attributs de l'ACI s'applique à l'attribut cible par "règle d'applicabilité pour les attributs".
- Noter que pour un ACI qui contient à la fois des listes grant est deny, la liste de permission grant peut ne pas être disponible pou cette règle d'applicabilité (voir plus bas)

Règles d'applicabilité pour les attributs

Si un attribut n'est pas spécifié comme partie de la ressource, alors cette règle ne s'applique pas. Si un attribut est spécifié, alors l'ACI en question s'applique si sa liste d'attribut est [all] ou si targetAttribute est explicitement mentionné dans la liste d'attribut de l'ACI.

Dans le cas où targetAttribute est un type d'attribut avec des options (ex : sn ;lang-en ;lang-uk), il est applicable si la liste d'attribut de l'ACI mentionne une forme moins spécifique de targetAttribute. Par exemple, si la liste contient "sn ;lang-en", alors la liste s'applique à l'attribut "sn ;lang-en ;lang-uk", mais pas l'attribut "sn".

Règles d'applicabilité pour les sujets

Appelons la portion sujet de l'ACI en question **aciSubject**. Pour déterminer si **aciSubject** s'applique au **requestorSubject** ou applique les règles suivantes :

1. L'ACI en question est un ACI grant. Alors l'ACI s'applique si les portions context et pureSubject de aciSubject s'applique, comme définis dans les règles d'applicabilité pour le contexte et les règles d'applicabilité pour pureSubject.
2. L'ACI en question est un ACI deny. Il y a 3 possibilité :
   a. La partie pureSubject s'applique (en accord avec les règles d'applicabilité pour pureSubject). Donc l'ACI s'applique au requestorSubject.
   b. La partie pureSubject ne s'applique pas. Donc l'ACI s'applique à tout requestorSubject avec un authnLevel inférieurs au authnLevel de l'ACI.
   c. Sinon l'ACI ne s'applique pas.
3. L'ACI en question est à la fois grant est deny. Il y à 3 possibilités :
   a. aciSubject s'applique quand évalué comme ACI grant. Les permissions grant et deny de l'ACI sont dispoible pour les règles d'applicabilité pour les attributs et les permissions.
   b. aciSubject ne s'applique pas comme ACI grant mais s'applique comme ACI deny. Les permissions deny de l'ACI sont disponibles pour les règles d'applicabilité pour les attributs et les permissions.
   c. aciSubject ne s'applique pas quand évalué soit à grant ou deny. L'ACI ne s'applique pas au requestorSubject.

Note : le mode deny avec authnLevel sert d'explication. Dans le cas où un ACI refuse d'accès à un sujet donné avec un authnLevel donné, alors naturellement il va refuser l'accès à ce subjet authentifié à un authnLevel ou supérieur. Similairement, un autre utilisateur authentifié au authnLevel ou supérieur, pour lequel la partie pureSubject ne s'applique pas, ne sera pas refusé les droits par cet ACI.

Le cas intéressant est un utilisateur authentifié à un niveau inférieur à authnLevel. Pour un tel utilisateur l'ACM considère que cet utilisateur n'a pas été prouvé au système, à un niveau de confiance suffisant, la partie pureSubject ne s'applique pas à lui, donc par sécurité, il se verra refuser les droits.

Donc si vous refusez l'accès à un authzId particulier avec authnLevel à "none", cet authzId aura l'accès refusé à tous les niveaux d'authentification, mais n'affectera pas d'autre demandeurs. D'une autre manière, si vous refusez l'accès à un authzId particulier avec un authnLevel "strong", celà va refuser l'accès à cet utilisateur quand il est authentifié "fort" et refusera l'accès à tout utilisateur authentifié avec des niveau d'authentification inférieurs.

Règles d'applicabilité pour pureSubject

Si aciSubject est de type anySubject, alors il s'applique au requestorSubject.

Si aciSubject est de type machineSubject, alors si ipAddress ou le nom dns de requestorSubject matche l'ipAddress ou la plage de nom dns dans aciSubject, alors l'ACI s'applique au requestorSubject si ce n'est pas un ACI deny ou la partie deny d'un ACI grant/deny. Un ACI grant ne s'applique jamais si le sujet est ipAddress : ou dns :. La note à la fin de "Précédence des sujets dans un scope" explique le raisonnement derrière cette règle.

Règles d'applicabilité pour le contexte

Le contexte d'un aciSubject s'applique au requestorSubject si authnLevel du requestorSubject est supérieur ou égal au authLevel spécifié dans la partie contexte de aciSubject.

Règles d'applicabilité pour idBasedSubject

si idBasedSubject est de type thisSubject, alors il s'applique au requestorSubject si authzId du requestorSubject est de type "dn" et est égal au DN de la ressource.

Si idBasedSubject est de type oneSubject, alors il s'applique au requestorSubject si authzId du requestorSubject est égal au authzId spécifié dans aciSubject.

Si idBasedSubject est de type setOfSubjects, alors il s'applique au requestorSubject si authzId du requestorSubject est définis dans le set spécifié dans aciSubject (ex : est dans ce groupe, rôle ou subtree). Le "Précédence des sujets dans un scope" inclus les règles pour déterminer le membership dans un setOfSubjects.

Règles de précédence

Les règles de précédences permetten à l'algorithme de décision de contrôle d'accès de déterminer quelles valeurs d'ACI ont précédence sur les autres.

Précédence des type de scope

1. Entry
2. Subtree

Précédence de position dans le DIT

Pour un DN sujet donné (incluant le niveau d'authentification) et un DN cible, subtreeACI plus bas dans l'arborescence prend précédence sur ceux plus haut.

Précédence de sujets dans un scope

1. ipAddress ou dns dans un ACI deny pour la partie deny d'un ACI grant/deny
2. authzId dn (authzId-dn :) ou authzId userid (authzId-u :)
3. this
4. role

Si le DN d'un rôle ou d'un groupe apparaît dans un rôle (ex : qui apparaît comme valeur de roleOccupant dans un organizationalRole), il est étendu récursivement. Pour déterminer la précédence, la collection de noms résultante de l'expansion est considéré venir d'un rôle sans regarder la source originale.

5. group

Si le DN d'un rôle ou d'un groupe apparaît dans un groupe, il est étendu récursivement. Pour déterminer la précédence, la collection de noms résultante de l'expansion est considéré venir d'un groupe sans regarder la source originale.

6. subtree

Les subtrees peuvent contenir les groupes et/ou des rôles. Ils peuvent être récursivement étendus. Pour déterminer la précédence, les membres des groupes ou occupants de rôles qui s'appliquent parce que le groupe ou le rôle est contenus dans un subtree sont considérés venir du subtree sans regarder la source originale.

7. public

La précédence de idAddress et des noms dns sont traités spécialement, et dépendent du type d'ACI. Typiquement les situations sont :

- Si une ACL dit d'autoriser sur la base de l'adresse IP mais refuse sur la base de certains autres attributs, alors, la méthode qu'on veut est "deny". ex : l'utilisateur est refusé, sans regarder où il réside.
- Si une ACL des de refuser sur la base d'adresse IP mais autorise sur la base d'autres attributs, la méthode qu'on veut est également "deny". ex : L'utilisateur est accepté, mais pas d'où il réside.

En plus, un grant à un ipAddress sans autre ACI applicable est très dangereux d'un point de vue sécuritaire, vu que cela donne accès a tout utilisateur qui a accès à cet ordinateur avec cette adresse donnée. Ainsi les sujets ipAddress et dns peuvent être utilisés seulement pour refuser les permissions, pas les accepter.

Spécifications de précédence d'attribut

Les contrôles d'accès spécifiant les attributs "[all]" ont une précédence plus faible que les listes spécifiques.

Précédence de grant/deny

Deny prend précédence sur grant.

Default

Deny est le défaut quand il n'y a pas d'information de contrôle d'accès ou quand l'évaluation des informations de contrôle d'accès ne fournis aucun résultat qui autorise d'accès.

Algorithme de décision de contrôle d'accès

L'algorithme prend en entrée les paramètres de décision définis plus haut et produit une décision grant ou deny. Dans le cas où on est intéressé dans le jeu de permission pour un jeu d'entrée et d'attributs (comme dans le cas de l'évaluation des droits effectifs), alors on doit appliquer l'algorithme pour chaque entrée/attribut et paire de permission qui nous intéresse. Naturellement les implémenteurs sont libre d'implémenter un algorithme qui produis la même décision sur une entrée et les valeurs d'ACI dans les DIT.

L'algorithme a 2 phases. D'abord, toutes les valeurs d'ACI potentiellement applicable sont triés en une liste ordonnée de jeu de valeurs d'ACI de la même précédence. Puis cette liste est scannée dans l'ordre pour trouver le jeu d'ACI qui va déterminer la décision d'accès.

Phase 1 : Ordonner les valeurs d'ACI potentiellement applicables

1. Détermine toutes les valeurs entryACI et subtreeACI qui s'appliquent au targetEntry, en accord avec les règles d'applicabilité pour les type de scope.
2. Trie ces ACI en une liste de jeux d'ACI de précédence égales, en accord avec les règles de précédence du type de scope. et la précédence de position dans le DIT.
3. Détermine quelles valeurs d'ACI collectée de l'étape 1 s'appliquent au requestorSubject en utilisant les règles d'applicabilité pour les sujets. Toutes les valeurs d'ACI qui ne s'appliquent pas à ce sujet sont éliminés.
4. La liste de jeux est triée en accord avec le type de sujet depuis les règles de précédence des sujets dans un scope.
5. Maintenant on split la liste en 2 listes conservant le même ordre relatif, une liste qui réfère au permissions au niveau entrée, et l'autre au niveau attributs.
6. Chaque jeu dans la liste de niveau attribut et ensuite divisée en une liste de jeux de précédence égale en accord avec la spécification de précédence d'attribut.

Note : à ce point on a 2 listes de jeux de valeurs d'ACI. Cette liste a été triée en accord avec les scope, la position, le sujet et (pour la liste au niveau attribut) les règles de spécification de précédence d'attribut.

Phase2 : Scanne les liste pour déterminer la décision d'accès :

1. Si permissionParameter est une permission au niveau entrée (donc le champ optionnel targetAttribute n'est pas spécifié), scanne la liste de niveau entrée dans l'ordre. Le premier jeu dans la liste qui contient un ACI qui s'applique au permissionParameter détermine la décision d'accès si un ACI dans le jeu autorise permissionParameter et aucun autre le refuse, alors l'accès est donné, sinon l'accès est refusé.

2. Si permissionParameter est une permission au niveau attribut, scanne la liste des jeux au niveau attribut dans l'ordre. Le premier jeu dans la liste qui contient un ACI qui s'applique au permissionParameter et s'applique à targetAttribute détermine la décision d'accès. Si un ACI dans le jeu autorise permissionParameter et aucun autre le refuse, l'accès est donné, sinon l'accès est refusé.

Exemple de précédence/héritage de décision d'accès

L'exemple dans cette section réfère à l'arborescence suivante :

```
___________dc=com
_____________|
____+-----+----------+
____|_________________________|
dc=tivoli___________________dc=sun
____|_________________________|
_cn=ellen___________________cn=rob
```

L'ACI à dc=com :

1. subtreeACI :grant :rsc#[all]#authnLevel :none :public :
2. subtreeACI :deny :rsc#userPassword,subtreeACI,entryACI,salary#authnLevel :none :public :
3. subtreeACI :grant :bvt#[entry]#authnLevel :none :public :

4. subtreeACI :grant :rscmow#[all]#authnLevel :strong :authzID-dn :cn=rob,dc=sun,dc=com
5. subtreeACI :grant :bvtugeinad#[entry]#authnLevel :strong :authzID-dn :cn=rob,dc=sun,dc=com

L'ACI à dc=tivoli,dc=com :

6. subtreeACI :grant :rsc ;deny :mow#[all]#authnLevel :strong :authzID-dn :cn=rob,dc=sun,dc=com
7. subtreeACI :deny :einad#[entry]#authnLevel :strong :authzID-dn :cn=rob,dc=sun,dc=com

L'ACI à cn=ellen,dc=tivoli,dc=com :

8. entryACI :grant :wo#[all]#authnLevel :strong :authz-ID-dn :cn=ellen,dc=tivoli,dc=com
9. entryACI :deny :wo#entryACI,subtreeACI,salary#authnLevel :strong :authz-ID-dn :cn=ellen,dc=tivoli,dc=com

Exemple 1

subject : ("cn=rob,dc=sun,dc=com", ipaddress, dns name, strong) :

resource : ("cn=ellen,dc=tivoli,dc=com", salary)

permission : "w"

Phase 1 : Touver tous les ACI applicables dans l'applicabilité des types de scope : {**1, 2, 3, 4, 5, 6, 7, 8, 9**}

Trier par précédence de type de scope et précédence de position dans le DIT : {**8, 9**}, {**6, 7**}, {**1, 2, 3, 4, 5**}

Détermine quels ACI sont applicable basé sur le sujet : {**6, 7**}, {**1, 2, 3, 4, 5**}

Trier par précédence de sujets dans le scope : {**6, 7**}, {**4, 5**}, {**1, 2, 3**}

Séparer les permissions applicables à l'entrée et ceux applicable aux attributs : **Entry :** {**7**}, {**5**}, {**3**}, **Attr :** {**6**}, {**4**}, {**1, 2**}

Trier les permissions applicable aux attributs par précédence de spécification d'attribut : **Entry :** {**7**}, {**5**}, {**3**}, **Attr :** {**6**}, {**4**}, {**2**}, {**1**}

Phase 2 : Vu que "w" est une permission attribut, recherche dans la liste Attr l'ACI 6 dans la première sous-liste mentionnant "w" et salary (via [all]) pour définir l'accès, qui est deny.

Exemple 2

subject : ("cn=rob,dc=sun,dc=com", ipaddress, dns name, limited) :

resource : ("cn=ellen,dc=tivoli,dc=com", salary)

permission : "w"

Phase 1 : Les ACI sont ordonnés dans les jeux suivant sont les sujets matchent le sujet : **Entry :** {**7**}, {**3**}, **Attr :** {**6**}, {**2**}, {**1**}

Phase 2 : Pour l'ACI 6 dans le premier jeu, qui matchait le sujet par la portion deny et limitée à < strong, les permissions disponibles sont "mow". Donc, cet ACI applique à "w" et salary (via [all]) et l'accès est deny.

Exemple 3

subject : ("cn=rob,dc=sun,dc=com", ipaddress, dns name, limited) :

resource : ("cn=ellen,dc=tivoli,dc=com", salary)

permission : "r"

Phase 1 : Les ACI sont ordonnés dans les jeux suivant sont les sujets matchent le sujet : **Entry :** {**7**}, {**3**}, **Attr :** {**6**}, {**2**}, {**1**}

Phase 2 : Vu que la portion grant de l'ACI 6 n'est pas active, le premier jeu qui contient un ACI qui s'applique à "r" et salary est {2}. Comme 2 deny l'accès, l'accès est refusé.

Exemple 4

subject : ("cn=rob,dc=sun,dc=com", ipaddress, dns name, limited) :

resource : ("cn=ellen,dc=tivoli,dc=com", cn)

permission : "r"

Phase 1 : Les ACI sont ordonnés dans les jeux suivant sont les sujets matchent le sujet : **Entry :** {**7**}, {**3**}, **Attr :** {**6**}, {**2**}, {**1**}

Phase 2 : Vu que la portion grant de l'ACI 6 n'est pas active, le premier jeu qui contient un ACI qui s'applique à "r" et cn est {1}. Vu que 1 grant l'accès, l'accès est donné.

Permissions requises pour chaque opération LDAP

Cette section définis les permissions requises pour chaque opération LDAP. Même si ces requis sont satisfaits, le serveur peut refuser l'opération dû à d'autres considérations de sécurité spécifique à l'implémentation. Par exemple, un serveur peut refuser de modifier une entrée parce que la base où réside l'entrée est en lecture seule. Un autre exemple peut être qui le contrôle d'accès LDAP a été spécifié sur l'attribut userPassword, mais un serveur peut refuser les modifications dû à certaines gouvernances de stratégie d'accès spécifique aux mots de passe.

Ici, on spécifie les droits requis par un utilisateur en effectuant un opération LDAP en termes de permissions LDAP spécifiés plus haut. Rappelons que les permissions "**a, d, e, i, n, b, v, t, u**" s'appliquent aux entrées et les permissions "**r, s, p, w, o, c, m, g**" s'appliquent aux attributs dans les entrées.

Les permissions requises pour les opérations étendues LDAP et les contrôles LDAP devraient être définis avec leurs spécification. Ces requis peuvent être définis en terme de ce modèle, par exemple en nécessitant une des permissions existante sur certaines entrées ou attributs particulier. Cette version du modèle de contrôle d'accès n'offre pas de mécanisme pour étendre le jeu de permissions ou la syntaxe d'ACI pour convenir aux opérations étendues ou aux contrôles.

Pour la suite, on assume que l'authorization ID de l'utilisateur faisant l'opération est authzId.

Opération Bind

Ce modèle ne nécessite pas de permission pour permettre de traiter une opération bind.

Opération Search

Rappelons que les paramètres de l'opération search sont :

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
 baseObject LDAPDN,
 scope ENUMERATED {
  baseObject (0),
  singleLevel (1),
  wholeSubtree (2) },
 derefAliases ENUMERATED {
  neverDerefAliases (0),
  derefInSearching (1),
  derefFindingBaseObj (2),
  derefAlways (3) },
 sizeLimit INTEGER (0 .. maxInt),
 timeLimit INTEGER (0 .. maxInt),
```

```
typesOnly BOOLEAN,
filter Filter,
attributes AttributeDescriptionList }
```

Supposons qu'un serveur traite une recherche de l'utilisateur authzId avec les paramètres comme précédent et traite l'entrée avec le dn candidateDN pour décider s'il peut le retourner ou non, alors les permissions requises par authzId qui doivent être évaluées sont les suivantes :

1. Permission "b" sur l'entrée candidateDN si l'entrée est dans le scope de recherche mais n'est pas l'entrée de base. Si cette permission n'est pas donnée alors le dn candidateDN ne doit pas être retourné ni un type d'attribut ni une valeur d'attribut de cette entrée. Note : la permission b est incluse pour permettre différents droits de découverte et de parcours à être assigné à différentes classes d'utilisateurs.
2. Permission "v" sur l'entrée candidateDN. Si cette permission n'est pas donnée alors de dn candidateDN ne doit pas être retourné, ni un type d'attribut ni une valeur d'attribut de cette entrée. La permission v est une permission de lecture au niveau de l'entrée.
3. Permission "p" ou "s" pour chaque attribut apparaissant dans un filtre de recherche de test de présence. "s" est requis sur les attributs apparaissant dans les tests de non-présence (equalityMatch, substrings, greaterOrEqual, lessOrEqual, present, approxMatch, extensibleMatch). La déclaration plus haut couvre le cas où les attributs sont évalués dans un extensibleMatch qui apparaît dans le filtre. Dans le cas où le champ dnAttributes de l'extensibleMatch est vrai il ne requière pas de vérification d'accès aux attributs du dn vu que l'accès est donné par la permission "v".

S'il y a un attribut dans un élément de filtre pour lequel la permission n'est pas donnée alors cet élément de filtre s'évalue à "Undefined" du three-valued-logic de X.511. Note : la motivation pour la permission "p" est que si vous avez un filtre de droits sur un attribut alors dans certains cas il pourrait être opérationnellement le même que la permission read. par ex. vous pouvez déterminer rapidement la valeur de l'attribut, et cela n'est pas forcément désirable. Par exemple, si le type de l'attribut est un entier alors avec les permissions du filtre vous pouvez rapidement déterminer la valeur en faisant une séquence de recherche en utilisant ">" et "<". Alors qu'avec la capacité de test de présence, vous pouvez empêcher ce genre de recherche, mais vous être capable de tester la présence de cet attribut.

4. Permission "r" pour chaque attribut dans la liste d'attribut AttributeDescriptionList (ou tous les attributs dans l'entrée candidateDN si AttributeDescriptionList est *) dont le type et/ou la valeur sera retournée. Note : la présence d'un attribut dans une entrée n'est jamais volontaire par le serveur si l'autorisation "r" lui est accordée, si un utilisateur peut déduire la présence d'un attribut avec "s" ou la permission "p" en utilisant un test de présence sur cet attribut dans le filtre de recherche. Note : bien que "r" et "s" sont typiquement donnés aux attributs on conserve les 2 permissions vu qu'il y a des cas où le distinction est utile. Une recherche de téléphone inversée est un exemple d'accès "r" mais pas "s".
5. Permission "t" à l'entrée candidateDN. Si cette permission n'est pas donnée alors le dn candidateDN ne doit pas être retourné. Si le serveur connais un alias pour l'entrée, cet alias peut être retourné à la place. Si aucun alias n'est disponible alors l'entrée candidateDN doit être omise du résultat de recherche.
6. Disclose on error pour l'opération de recherche. S'il n'y a pas d'entrée dans le scope de recherche qui satisfait l'item 1 (voir les droit sur l'entrée candidate) et l'item 2 (permission read sur l'entrée) alors la permission "u" sur l'entrée de base doit être évaluée. Si "u" n'est pas donné alors l'opération doit échouer avec un "no such object error" et le matchedDN de LDAPResult doit être définis à "". Si "u" est donné au baseObject alors le jeu vide de résultat est retourné.

Protéger le nommage des attributs du DN

Protéger le nommage d'attribut dans le dn d'une entrée présente un problème pour le contrôle d'accès. Le problème est que si l'accès est donné au dn d'une entrée donnée, alors via le nommage d'attribut que ce dn contient, l'accès est également donné aux valeurs d'attribut dans les autres entrées. En plus, la connaissance de l'existence les entrées parent d'une entrée donnée est également fournie par le dn de l'entrée.

Opération modify

Rappelons que les paramètres de l'opération Modify sont :

```
ModifyRequest ::= [APPLICATION 6] SEQUENCE {
 object LDAPDN,
 modification SEQUENCE OF SEQUENCE {
  operation ENUMERATED {
   add (0),
   delete (1),
   replace (2) },
  modification AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
 type AttributeDescription,
 vals SET OF AttributeValue }
```

Les permissions requises par authzId qui doivent être évaluées sont :

1. Permissions "w" pour chaque attribut à ajouter à l'object
2. Permission "o" pour chaque attribut pour lequel une valeur est supprimée de l'objet
3. permission "o" et "w" à chaque attribut remplacé dans l'objet.

Opération Add

Rappelons que les paramètres de l'opération Add sont :

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
 entry LDAPDN,
 attributes AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
 type AttributeDescription,
 vals SET OF AttributeValue }
```

1. Permission "a" au parent de l'entrée
2. Permission "m" au parent de l'entrée pour chaque attribut à ajouter à l'entrée

Opération Delete

Rappelons que les paramètres de l'opération Delete sont :

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

1. Permission "d" sur l'entrée

Note : on pourrait ajouter la permission "o" pour autoriser l'opération, mais l'expérience a montré que cette permission additionnelle n'est pas aussi utile que ça, et X.500 ne nécessite que la permission "d"

Opération Modify DN

Rappelons que les paramètres de l'opération Modify DN sont :

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
 entry LDAPDN,
 newrdn RelativeLDAPDN,
 deleteoldrdn BOOLEAN,
 newSuperior [0] LDAPDN OPTIONAL }
```

Les variantes de l'opération ModifyDN sont listés ci-dessous. Les combinaisons des permissions write, obliterate, import, export et renameDN sont nécessaire pour chaque variante.

1. Renommer une entrée en déplaçant le flag RDN entrée 2 valeurs d'attributs existant, sans altérer de valeur d'attribut. Les permissions nécessaires sont renameDN.
2. Renommer une entrée en ajoutant une nouvelle valeur d'attribut, les permissions nécessaires sont renameDN et write
3. Renommer une entrée en utilisant une valeur d'attribut existant et en supprimant la valeur d'attribut courante. Les permissions nécessaires sont renameDN, write et obliterate.
5. Renommer une entrée en ajoutant une nouvelle valeur d'attribut et en supprimant la valeur courante. Les permissions nécessaires sont renameDN, write et obliterate.
5. Déplacer une entrée à un nouvel endroit dans le DIT, en gardant son RDN. Les permissions nécessaires sont import et export
6. Déplacer une entrée à un nouvel endroit couplé avec 1. Les permissions nécessaires sont import, export et renameDN
7. Déplacer une entrée couplé avec 2. Les permissions nécessaires sont import, export, renameDN, et write.
8. Déplacer une entrée couplé avec 3. Les permissions nécessaires sont import, export, renameDN, et obliterate.
9. Déplacer une entrée couplé avec 4. Les permissions nécessaires sont import, export, renameDN, write et obliterate.

Pour un cas donné, si les permissions requises sont données, alors l'opération est permise par le modèle de contrôle d'accès. Si, pour un cas donné, les permissions ne sont pas données, alors l'opération doit échouer. Si le contrôle d'accès échoue à cause d'une permission d'attribut ou sur l'entrée manquante, alors si "u" est donné à l'entrée, le code d'erreur et matchedDN peuvent être retournés. Si "u" n'est pas donné à l'entrée, alors nosuchObject doit être retourné et matchedDN définis à "". Une logique similaire s'applique si l'échec du contrôle d'accès est dûe a une permission manquante sur newSuperior.

Opération Compare

Rappelons que les paramètres de l'opération Compare sont :

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
 entry LDAPDN,
 ava AttributeValueAssertion }
```

1. Permission "c" sur l'attribut dans l'entrée sur lequel la comparaison est faite.

Opération Abandon

Rappelons que les paramètres de l'opération Abandon sont :

```
AbandonRequest ::= [APPLICATION 16] MessageID
```

authzId a toujours les droits d'envoyer l'opération Abandon pour une opération précédemment initiée.

Opération Extended

Rappelons que les paramètres de l'opération Exended sont :

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
 requestName [0] LDAPOID,
 requestValue [1] OCTET STRING OPTIONAL }
```

L'accès requis pour une opération étendue est au delà du scope de ce document. L'accès requis sera normalement définis par l'implémenteur de la requête étendue.

Permissions requises pour manipuler les alias et les références

L'utilisation d'alias et le réferrants font partie de LDAPv3. Cependant, ils ne sont pas particulièrement bien définis. Les objets et attributs alias sont définis dans la rfc2256 comme dérivé de X.500, mais ne définis pas leur sémantique. X.500 définis les sémantiques des alias avec le respect du contrôle d'accès. On définis ce principe dans LDAPv3 basé sur X.511. Les référrants sont définis dans X.500, mais ne définis par leur sémantique avec le respect des contrôles d'accès.

ACI Distribution

Actuellement il n'y a pas de standard LDAP définissant comment distribuer les données d'annuaire entre les serveur LDAP. En conséquence ce modèle ne peut pas spécifier pleinement le fonctionnement du modèle de contrôle d'accès dans un environnement distribué. Le cas de distribution via des référants est traité plus bas. Dans le cas du chaînage (où un serveur LDAP renvoie une requête à un autre pour le compte d'un client) alors le fonctionnement du modèle de contrôle d'accès est spécifique à ce serveur. Similairement, la manière dont un serveur détermine si le chaînage d'une opération est permise est spécifique au serveur. Par exemple, l'implémentation peut choisir de regarder le contexte de nommage local et le contexte de nommage subordonné distant comme aire de contrôle d'accès spécifique séparé, ou peut regarder le DIT comme une aire spécifique de contrôle d'accès et d'implémenter les mécanismes pour propager les aci entre les 2 serveurs. Ceci est hors du scope de ce modèle.

Alias

Il y a 2 choses pour protéger avec les alias : le vrai nom des objets aliasés et l'emplacement du serveur le maintenant. Si le dé-référencement d'alias est requis dans le processus de localisation d'une entrée cible, aucune permissions spécifique n'est nécessaire pour le dé-référencement de l'alias. Le contrôle d'accès est forcé sur l'objet pointé par l'alias. Si le dé-référencement d'alias résulte en un continuationReference (ex : depuis une opération de recherche), alors la permission browse est requise sur l'alias et la permission read est requise sur l'attribut.

Référants

Si un référant doit être suivi, aucune permission spécifique n'est nécessaire pour le client ldap pour suivre le référant. Le contrôle d'accès est forcé sur l'objet référencé. Si un référant est retourné, alors browse est requis sur l'entrée et read est requis sur l'attribut contenant le référant. Si le serveur implémente un référant par défaut, il n'y a pas de permission spécial.

Contrôler l'accès aux ACI

Les attributs entryACI et subtreeACI sont utilisé pour spécifier le contrôle pour qui a la permission de définir/changer l'ACI. Les attributs entryACI et subtreeACI sont juste un autre attribut décrit avec un jeu de droits et permissions, et des sujets comme valeur de entryACI et subtreeACI. Si la stratégie pour contrôler entryACI et subtreeACI n'est pas spécifiée pour les objets dans l'arborescence, le mode est définis par l'implémentation. Par exemple, si aucun objet dans l'arborescence ne définis l'accès pour entryACI/subtreeACI dans les attributs entryACI/subtreeACI, alors le serveur pourrai simplement affirmer que le root DN est considéré comme propriétaire des stratégies pour tous les objets.

Exemples d'ACI

Noter que dans les exemples, la forme "OID.<attrname>" réfère à l'OID sous la forme décimale pour l'attribut <attrname>. Cette notation est utilisée uniquement pour les exemples.

Définitions d'attribut

L'exemple suivant montre l'accès requis pour contrôler l'accès aux attributs entryACI et subtreeACI. Le premier exemple montre le contrôle sur une entrée individuelle et ses attributs. Le second exemple montre le contrôle d'accès sur un subtree. Le authnLevel est définis pour être raisonnablement sécurisé.

entryACI : grant :rwo#OID.entryACI#authnLevel :limited :role :cn=aciAdmin
subtreeACI : grant :rwo#OID.subtreeACI#authnLevel :limited :role :cn=aciAdmin
L'exemple suivant montre un attribut subtreeACI où un groupe "cn=DeptXYZ,c=US" a les permissions read,search et compare sur l'attribut attr1. Les permissions s'appliquent à tout le subtree sous le nœud contenant l'aci.

subtreeACI : grant ;rsc#OID.attr1#authnLevel :weak :group :cn=Dept XYZ,c=US
L'exemple suivant montre un attribut ACI où un rôle "cn=SysAdmins,o=Company" a les permissions browseDN, viewEntry, et returnDN pour les objets sous ce nœud. Le rôle a également les droits read, search, et compare sur l'attribut attr2 et read, search, compare, write et obliterate sur attr3.

subtreeACI : grant :bvt#[entry]#authnLevel :weak :role :cn=SysAdmins,o=Company
subtreeACI : grant :rsc#OID.attr2#authnLevel :weak :role :cn=SysAdmins,o=Company
subtreeACI : grant :rscwo#OID.attr3#authnLevel :weak :role :cn=SysAdmins,o=Company

Modifier les valeurs entryACI et subtreeACI

Modify-Replace fonctionne comme définis dans l'opération ldap modify. Si la valeur d'attribut n'existe pas, il créé la valeur. Si l'attribut existe, il remplace la valeur. Si les valeurs entryACI/subtreeACI sont remplacées, toutes les valeurs entryACI/subtreeACI sont remplacées. Modifier les attributs entryACI/subtreeACI nécessite d'avoir les permissions "w" et "o" sur entryACI/subtreeACI. Les exemples dans cette section assument que vous avez les accès pour contrôler entryACI/subtreeACI.

Un subtreeACI donnée pour une entrée :
subtreeACI : deny :rw#[all]#authnLevel :weak :group :cn=Dept ABC
subtreeACI : grant :r#OID.attr1#authnLevel :weak :group :cn=Dept XYZ
Effectue le changement suivant :
dn : cn=someEntry
changetype : modify
replace : subtreeACI
subtreeACI : grant :rw#[all]#authnLevel :weak :group :cn=Dept LMN
L'ACI résultante est :
subtreeACI : grant :rw#[all]#authnLevel :weak :group :cn=Dept LMN
Les valeurs subtreeACI pour Dept XYZ et ABC sont perdues durant le remplacement

Durant un ldapmodify-add, si l'ACI n'existe pas, créé l'ACI avec les valeurs spécifique entryACI/subtreeACI. Si l'ACI existe, ajoute les valeurs spécifiées à l'entryACI/subtreeACI donné.

Par exemple un ACI donné :
subtreeACI : grant :rw#[all]#authnLevel :weak :group :cn=Dept XYZ
Avec une modification :
dn : cn=someEntry
changetype : modify
add : subtreeACI
subtreeACI : grant :r#OID.attr1#authnLevel :weak :group :cn=Dept XYZ
Va donner un ACI multi-valué :
subtreeACI : grant :rw#[all]#authnLevel :weak :group :cn=Dept XYZ
subtreeACI : grant :r#OID.attr1#authnLevel :weak :group :cn=Dept XYZ
Pour supprimer une valeur d'ACI particulière, utiliser la syntaxe ldapmodify-delete. Un ACI donné de :
subtreeACI : grant :rw#[all]#authnLevel :weak :group :cn=Dept XYZ
subtreeACI : grant :r#OID.attr1#authnLevel :weak :group :cn=Dept XYZ
dn : cn = some Entry
changetype : modify
delete : subtreeACI
subtreeACI : grant :r#OID.attr1#authnLevel :weak :group :cn=Dept XYZ
Va donner un ACI restant :
subtreeACI : grant :rw#[all]#authnLevel :weak :group :cn=Dept XYZ
Avec l'opération ldapmodify-delete, toute l'acl peut être supprimée en spécifiant :
dn : cn = some Entry
changetype : modify
delete : entryACI
dn : cn = some Entry
changetype : modify
delete : subtreeACI
Dans ce cas, l'entrée va hériter son ACI d'autres nœuds dans l'arborescence. Similairement, si toutes les valeurs d'entryACI et subtreeACI sont supprimées, alors les informations de contrôle d'accès pour cette entrée sont définis par le modèle d'héritage.

Évaluation

Ces exemples assument que les entrées ACI listées dans chaque exemple sont les seuls ACI qui s'appliquent à l'entrée en question. On assume cn=jsmith est membre du groupe cn=G1 et du groupe cn=G2.

Exemple 1 :
dn : o=XYZ, c=US
subtreeACI : grant :r#attr2#authnLevel :weak :group :cn=G1,ou=ABC,o=XYZ,c=US
subtreeACI : grant :w#attr2#authnLevel :weak :group :cn=G2,ou=ABC,o=XYZ,c=US
cn=jsmith a les droits rw sur attr2 ; l'ACI est combinées parce que les sujets (groupes) ont la même précédence.
Exemple 2 :
dn : o=XYZ, c=US
subtreeACI : grant :rw#OID.attr3#authnLevel :weak :group :cn=G1,ou=ABC,o=XYZ,c=US
subtreeACI : deny :w#OID.attr3#authnLevel :weak :group :cn=G2,ou=ABC,o=XYZ,c=US
cn=jsmith a l'accès read sur attr3 ; write est refusé puisque deny a précédence sur grant.
Exemple 3 :
dn : o=XYZ, c=US
subtreeACI : grant :m#OID.attr5#authnLevel :weak :authzID-dn :cn=jsmith,o=ABC,c=US
SubtreeACI : grant :m#OID.cn#authnLevel :weak :authzID-dn :cn=jsmith,o=ABC,c=US
subtreeACI : grant :m#OID.sn#authnLevel :weak :authzID-dn :cn=jsmith,o=ABC,c=US
subtreeACI : grant :a#[entry]#authnLevel :weak :authzID-dn :#cn=jsmith,o=ABC,c=US
cn=jsmith a le droit make sur attr5, cn et sn, et add sur l'entrée qui lui permet de créer un nouvel objet, cn=New,oXYZ,c=US avec des valeurs pour attr5, cn et sn. Cet exemple illustre comment la permission "m" peut être utilisée pour limiter les attributs qui peuvent être

créés sur une nouvelle entrée.
Exemple 4 :
dn : c=US
subtreeACI : grant :m#[all]#authnLevel :weak :subtree :c=US
dn : o=XYZ, c=US
subtreeACI : grant :a#[entry]#authnLevel :weak :authzID-dn :cn=jsmith,o=ABC,c=US
cn=jsmith a le droit make sur tous les attributs et add sur l'entrée. Il y'a suffisamment de permissions pour créer un nouvel objet, cn=New,oXYZ,c=US avec les valeurs pour n'importe quel attribut. Pour les administrateurs qui ne souhaitent par limiter les attributs qui peuvent être créés sur les nouvelles entrées, cet exemple montre comment un simple ldapACI au niveau du domaine résout le problème.
Exemple 5 :
dn : dc=com,dc=demo
subtreeACI : grant :rw#description ;lang-en#authnLevel :weak :authzID-dn :cn=rvh,dc=att,dc=com
subtreeACI : grant :rw#description ;lang-en,description ;lang-fr#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
Dans cet exemple, cn=rvh,dc=att,dc=com a l'accès "rw" à la langue anglaise de l'attribut Description sur les entrée sous dc=com,dc=demo. cn=rob,dc=sun,dc=com a les droits "rw" sur les versions française et anglaise de l'attribut Description. L'exemple démontre que "Attribute Descriptions", pas seulement "Attribute Types", peut être utilisé dans le champ attr d'un ACI.

ModDN

Il y a de nombreuses actions différentes qui peuvent se produire quand la commande modDN est utilisée. La suite illustre les permissions nécessaires pour exécuter chaque scénario. Pour tous les exemples, on assume que le rôle cn=Admins fonctionne avec l'entrée suivante :

dn : cn=personA,o=Company
cn : personA
cn : FirstName
sn : LastName
objectclass : person
Exemples 1 :
Effectuer un modRDN uniquement, utilisant une valeur d'attribut existante. Dans ce cas, on effectue un modRDN et on renomme cn=personA,o=Company en cn=FirstName,o=Company. La valeur d'entryACI pour cette entrée doit donner la permission rename sur l'entrée.
entryACI : grant :n#[entry]#authnLevel :weak :role :cn=Admin
Exemple2 :
Effectuer un modRDN et ajouter une nouvelle valeur d'attribut. Dans ce cas on renomme cn=personA,o=Company en cn=newFirstName,o=Company. La valeur entryACI doit donner la permission rename sur l'entrée et w sur l'attribut cn.
entryACI : grant :n#[entry]#authnLevel :weak :role :cn=Admin
entryACI : grant :w#cn#authnLevel :weak :role :cn=Admin
Exemple 3 :
Effectuer un modRDN, en utilisant un attribut existant, mais en supprimant l'ancienne valeur RDN. On renomme cn=personA,o=Company en cn=FirstName,o=Company avec le flag deleteOldRdn à true. On doit avoir les permissions de renommer l'entrée, et de supprimer une valeur cn
entryACI : grant :n#[entry]#authnLevel :weak :role :cn=Admin
entryACI : grant :o#OID.cn#authnLevel :weak :role :cn=Admin
Exemple 4 :
Effectuer un modRDN, en utilisant une nouvelle valeur d'attribut, et en supprimant l'ancienne valeur. dans ce cas on renomme cn=personA,o=Company en cn=newFirstName,o=Company avec le flag deleteOldRdn. On doit avoir les permissions de renommer l'entrée, et de supprimer et écrire l'attribut cn
entryACI : grant :n#[entry]#authnLevel :weak :role :cn=Admin
entryACI : grant :w,o#OID.cn#authnLevel :weak :role :cn=Admin
Exemple 5 :
On veut changer l'emplacement de l'entrée et conserver le même RDN. Dans ce cas, on déplace cn=personA,o=Company vers cn=personA,o=CompanyB. On doit avoir les permissions d'export sur l'entrée originale, et import dur le nouvel objet supérieur.
l'entryACI de cn=personA,o=Company est :
entryACI : grant :e#[entry]#authnLevel :weak :role :cn=Admin
et l'entryACI de o=CompanyB :
entryACI : grant :i#[entry]#authnLevel :weak :role :cn=Admin
Exemple 6 :

On veut changer l'emplacement de l'entrée et changer la valeur RDN à une valeur existante. Dans ce cas on déplace cn=personA,o=Company vers cn=FirstName,o=CompanyB. On doit avoir les premissions rename et export sur l'objet, et import sur le nouvel objet supérieur.

l'entryACI de cn=personA,o=Company est :

entryACI : grant :en#[entry]#authnLevel :weak :role :cn=Admin

et l'entryACI de o=CompanyB :

entryACI : grant :i#[entry]#authnLevel :weak :role :cn=Admin

Exemple 7 :

Changer l'emplacement de l'entrée et changer le RDN avec une nouvelle valeur. Dans ce cas, on déplace cn=personA,o=Company vers cn=newFirstName,o=CompanyB. On doit avoir les droits rename et export sur l'entrée originale, write sur cn et import sur le nouveau supérieur.

l'entryACI de cn=personA,o=Company est :

entryACI : grant :en#[entry]#authnLevel :weak :role :cn=Admin

entryACI : grant :w#OID.cn#authnLevel :weak :role :cn=Admin

et l'entryACI de o=CompanyB :

entryACI : grant :i#[entry]#authnLevel :weak :role :cn=Admin

Exemple 8 :

Changer l'emplacement de l'entrée et changer de RDN avec une valeur existante, en supprimant l'ancienne valeur. Dans ce cas, on déplace cn=personA,o=Company vers cn=FirstName,o=CompanyB. On doit avoir les droits rename et export sur l'entrée originale, delete sur cn et import sur le nouveau supérieur.

l'entryACI de cn=personA,o=Company est :

entryACI : grant :en#[entry]#authnLevel :weak :role :cn=Admin

entryACI : grant :o#cn#authnLevel :weak :role :cn=Admin

et l'entryACI de o=CompanyB :

entryACI : grant :i#[entry]#authnLevel :weak :role :cn=Admin

Exemple 9 :

Changer l'emplacement de l'entrée et changer de RDN avec une nouvelle valeur d'attribut, en supprimant l'ancienne valeur. Dans ce cas, on déplace cn=personA,o=Company vers cn=newFirstName,o=CompanyB. On doit avoir les droits rename et export sur l'objet original, write et delete sur cn, et import sur le nouveau supérieur

l'entryACI de cn=personA,o=Company est :

entryACI : grant :en#[entry]#authnLevel :weak :role :cn=Admin

entryACI : grant :wo#OID.cn#authnLevel :weak :role :cn=Admin

et l'entryACI de o=CompanyB :

entryACI : grant :i#[entry]#authnLevel :weak :role :cn=Admin

Intéraction entre les ACI

Ces exemples montrent comment les ACI dans différentes parties de l'arborescence intéragissent. Les exemples avec divers authnLevel sont donnés dans la section suivante.

Les exemples réfèrent à ce fragment de l'arborescence :

```
__________dc=com
___________|
___+-----+----------+
___|_______________________|
dc=tivoli_________________dc=sun
___|_______________________|
cn=ellen__________________cn=rob
```

Exemple 1 : Si l'ACI est le suivant :

À dc=com : subtreeACI :grant :rw#[all]#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com

À dc=tivoli,dc=com : subtreeACI :grant :r#[all]#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com

Alors les droits effectifs de cn=rob,dc=sun,dc=com sur tous les attributs de l'objet cn=ellen,dc=tivoli,dc=com sont "rw". L'ACI au niveau de dc=tivoli,dc=com est redondante.

Exemple 2 : Si l'ACI est le suivant :

À dc=com : subtreeACI :grant :r#[all]# authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com

À dc=tivoli,dc=com : subtreeACI :grant :w#OID.uid#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits "r" sur tous les attributs de l'objet cn=ellen,dc=tivoli,dc=com, et les droits "rw" sur l'uid de ce même objet. Également, cn=rob,dc=sun,dc=com a les droits "r" sur tous les attributs de cn=rob,dc=sun,dc=com
Exemple 3 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#[all]#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
À dc=tivoli,dc=com : subtreeACI :deny :w#[all]#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits "r" (mais pas "w") sur tous les attributs de cn=ellen,dc=tivoli,dc=com et "rw" sur tous les attributs de cn=rob,dc=sun,dc=com.
Exemple 4 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :r#OID.uid#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
À dc=tivoli,dc=com : subtreeACI :grant :w#OID.sn#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits "r" sur uid et "w" sur sn de cn=ellen,dc=tivoli,dc=com
Exemple 5 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :r#OID.uid#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
À cn=rob,dc=sun,dc=com : entryACI :grant :rw#[all]#authnLevel :weak :this :
Alors cn=rob,dc=sun,dc=com a des droits "rw" sur tous les attributs de cn=rob,dc=sun,dc=com.
Exemple 6 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#uid#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
À dc=tivoli,dc=com : subtreeACI :deny :w#uid#authnLevel :weak :subtree :dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits "r" sur l'uid de cn=ellen,dc=tivoli,dc=com. En vérifiant la permission "w", dc=tivoli,dc=com est inférieur à dc=com, donc sont subtreeACI a précédence.
Exemple 7 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#OID.uid#authnLevel :weak :authzID-dn :cn=rob,dc=sun,dc=com
À dc=com : subtreeACI :deny :w#OID.uid#authnLevel :weak :subtree :dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits "rw" sur uid de cn=ellen,dc=tivoli,dc=com. en vérifiant la permission "w", les 2 subtreeACI sont au même niveau dans l'arborescence et le type de sujet dn a précédence sur le type subtree, donc le premier aci a précédence sur le second.
Exemple 8 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#OID.uid#authnLevel :weak :subtree :dc=sun,dc=com
À dc=com : subtreeACI :deny :w#OID.uid#authnLevel :weak :subtree :dc=com
Alors cn=rob,dc=sun,dc=com a les droits "r" sur uid de cn=ellen,dc=tivoli,dc=com. En vérifiant la permission "w", les 2 subtreeACI sont au même niveau et ont le même type de sujet, donc deny a précédence sur grant dans le facteur de décision.
Exemple 9 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#OID.uid#authnLevel :weak :subtree :dc=sun,dc=com
À dc=com : subtreeACI :deny :w#[all]#authnLevel :weak :subtree :dc=com
Alors cn=rob,dc=sun,dc=com a les droits "rw" sur uid de cn=ellen,dc=tivoli,dc=com. En vérifiant la permission "w", les 2 subtreeACI sont au même niveau et ont le même type de sujet, donc la précédence d'une liste spécifique d'attributs sur [all] est le facteur de décision.

Utilisation de ipAddress dans les ACI

Exemple 1 : Si l'ACI est le suivant :
À dc=com : subtreeACI : deny :adeinbvtug#[entry]#authnLevel :strong :ipAddress :10.0.0.0-10.255.255.255
subtreeACI : deny :rwospcm#[all]#authnLevel :strong :ipAddress :10.0.0.0-10.255.255.255
subtreeACI : grant :rscp#[all]#authnLevel :none :public :
subtreeACI : grant :btv#[entry]#authnLevel :none :public :
Alors tout utilisateur n'a de permission s'ils se connectent depuis un réseau 10/. S'ils se connectent depuis un autre réseau, ils auront les permissions "rscp" pour tous les attributs, et "btv" pour toutes les entrées dans dc=com.
Exemple 2 : Si l'ACI est le suivant :
À dc=com : subtreeACI : grant :adeinbvtug#[entry]#authnLevel :weak :ipAddress :10.0.0.0-10.255.255.255
subtreeACI : grant :rspwocm#[all]#authnLevel :weak :ipAddress :10.0.0.0-10.255.255.255
Cela n'aura pas d'effet. Alors que cela semble donner un accès total aux utilisateurs dans le réseau 10/, les règles spéciales les idAddress et dns comme sujet les rendes utiles seulement pour les ACI deny. Les effets d'un grant sur une plage réseau mal formé ou un DNS wildcardé peut être très sérieux.
Un Administrateur qui veut vraiment donner un accès total à tout le monde dans 10/ devra spécifier :
À dc=com : subtreeACI :
grant :adeinbvtug#[entry]#authnLevel :weak :public :subtreeACI :deny :adeinbvtug#[entry]#authnLevel :strong :ipAddress :0.0.0.0-9.255.255.255,11.0.0.0-255.255.255.255
subtreeACI : grant :rspwocm#[all]#authnLevel :weak :public :

subtreeACI : deny :rspwocm#[all]#authnLevel :strong :ipAddress :0.0.0.0-9.255.255.255,11.0.0.0-255.255.255.255

Utilisation de authnLevel dans les ACI

Exemple 1 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#OID.sn#authnLevel :strong :authzID-dn :cn=rob,dc=sun,dc=com
À dc=tivoli,dc=com : subtreeACI :grant :r#OID.sn#authnLevel :limited :authzID-dn :cn=rob,dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits effectifs "rw" sur l'attribut sn de l'objet cn=ellen,dc=tivoli,dc=com si le bind pour cette session a utilisé une authentification forte. Si le bind a utilisé une authentification limitée, il n'a que le droit "r". Si le bind pour cette session a uilisé une authentification faible, ou aucune authentification, il n'a aucun droit.
Exemple 2 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#sn#authnLevel :limited :subtree :dc=sun,dc=com
À dc=tivoli,dc=com :subtreeACI :grant :c ;deny :w#sn#authnLevel :strong :authzID-dn :cn=rob,dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits effectifs "rc" sur l'attribut sn de l'objet cn=ellen,dc=tivoli,dc=com si le bind était fort. La permission "r" vient du fait que la partie grant du premier ACI s'applique au bind limité et supérieur. La partie deny du 2ème ACI s'applique aux cas où authnLevel est inférieur à "strong", donc il a précédence sur la permission "w" dans le premier ACI.
Exemple 3 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rs#sn#authnLevel :none :public
À dc=com : subtreeACI :grant :w#sn#authnLevel :strong :subtree :cn=rob,dc=sun,dc=com
Alors cn=rob,dc=sun,dc=com a les droits effectifs "rsw" sur l'attribut sn de l'objet cn=ellen,dc=tivoli,dc=com si le bind a utilisé une authentification forte, et "rs" si le bind a utilisé une autre forme d'authentification. Le grant dans le premier ACI s'applique aux bind au niveau "none" et supérieur.
Exemple 4 : Si l'ACI est le suivant :
Au root : subtreeACI :grant :ps#[all]#authnLevel :none :public :subtreeACI :grant :cr#[all]#authnLevel :weak :subtree :
Alors tout utilisateur (incluant les anonymes) ont les droits "ps" sur toutes les entrées sur le serveur, et tout utilisateur avec un ID sur le serveur dont le bind a utilisé weak ou mieux a les permissions "pscr".
Exemple 5 : Si l'ACI est le suivant :
À dc=com : subtreeACI :grant :rw#[all]#authnLevel :limited :cn=ellen,dc=tivoli,dc=com
À dc=tivoli,dc=com : subtreeACI :deny :w#[all]#authnLevel :strong :cn=rob,dc=sun,dc=com
Alors si le bind était fort, cn=ellen,dc=tivoli,dc=com a les permissions "rw" sur tous les attributs de l'objet cn=ellen,dc=tivoli,dc=com, et les permissions "rw" sur tous les attributs de cn=rob,dc=sun,dc=com. Si le bind est limité, cn=ellen,dc=tivoli,dc=com n'a plus que le droit "r" sur lui-même.
C'est une conséquence de la manière dont deny est traité avec authnLevel. Vu que cn=rob,dc=sun,dc=com n'a pas "w" quand il s'authentifie en strong, tous les utilisateurs n'auront pas ce droit quand ils s'authentifient à un niveau inférieur.

Contrôle GetEffectiveRights

Les contrôles d'aci fournissent une manière de manipuler les informations de contrôle d'accès, en conjonction avec une opération ldap. Un contrôle LDAP est définis. Ce contrôle permet de récupérer les informations de contrôle d'accès. Le contrôle est **GetEffectiveRights**. Son but est de permettre à un administrateur ou une application de demander au serveur les droits d'un autre utilisateur de l'annuaire. Cela permet à un administrateur de vérifier les droits d'un utilisateur, ou une application peut proposer à un utilisateur les attributs sur lequel il a des droits de modification ou de lecture.

Request Control

Ce contrôle peut être inclus uniquement dans un message ldap_search. Le controlValue est un OCTET STRING, dont la valeur est la valeur encodée BER :

```
GetEffectiveRightsRequest ::= SEQUENCE{
 gerSubject [0] GERSubject
}

GERSubject ::= SEQUENCE {
```

```
 gerOneSubject [0] OneSubject - from 4.1.2 , OPTIONAL
 germachineSubject [1] GERMachineSubject,
 gerAuthnLEvel [2] AuthnLevel, - from 4.1.2
}

GERMachineSubject ::= SEQUENCE{
 gerOneIPAddress [0] IPAddress, - from 4.1.2
 gerOneDNSName [1] DomainName - from 4.1.2
}
```

Le getEffectiveRightsRequest spécifie un sujet, gerSubject, sur quelles informations de contrôle d'accès sont demandées. Le contrôle demande au serveur d'évaluer et retourner les droits au niveau de l'entrée possédée par le gerSubject pour chaque entrée qui est retournée dans le résultat de recherche et pour chaque attribut spécifiquement demandé. Le serveur va utiliser l'algorithme de décision d'accès pour déterminer les droits effectifs.

Response Control

Ce contrôle est inclus dans un message de réponse ldap_search. Le contrôleValue est un OCTET STRING dont la valeur est encodée BER :

```
GetEffectiveRightsResponse ::= {
 result ENUMERATED {
  success (0),
  operationsError (1),
  unavailableCriticalExtension (12),
  noSuchAttribute (16),
  undefinedAttributeType (17),
  invalidAttributeSyntax (21),
  insufficientRights (50),
  unavailable (52),
  unwillingToPerform (53),
  other (80)
 }
}
```

Les droits effectifs sont retournés avec chaque entrée retournée par le résultat de la recherhe. Le contrôle de réponse pour ldap_search est :

```
PartialEffectiveRightsList ::= SEQUENCE {
 entryLevelRights [0] EffectiveRights,
 attributeLevelRights [1] AttributeLevelRights
}

EffectiveRights ::= CHOICE {
 rights [0] Permissions - from 4.1.2,
 noRights [1] NULL,
 errorEvaluatingRights [2] GerError
}

GerError ::= ENUMERATED
 {generalError(0),insufficientAccess(1)}

 AttributeLevelRights ::= SEQUENCE OF {
 attr [0] SEQUENCE OF Attribute,
 rights [1] EffectiveRights
}
```

Pour une entrée donnée, le champ entryLevelRights du contrôle de réponse contient les droits effectifs au niveau de l'entrée qui gerSubject a sur cette entrée. Le champ attributeLevelRights contient la liste des attributs et les droits effectifs que gerSubject a pour chacun de ces attributs. La liste des attributs consiste de ceux retournés dans l'opération de recherche et les attributs explicitement demandés. Un attribut explicitement demandé dans une recherche peut ne pas être retournée parce que l'entrée n'est pas présent dans l'entrée, mais on peut être intéressé par les permissions sur cet attribut.

Le contrôle retourne les permissions que gerSubject a sur l'entrée donnée et ses attributs. Pour déterminer si ces permissions suffisent pour permettre à gerSubject d'effectuer une opération LDAP donnée sur l'entrée, le demandeur va déterminer si ces permissions satisfont les permissions requises pour cette opération LDAP. Noter que dans le cas où les permissions ne sont pas suffisantes pour une acction, ce contrôle ne permet pas de déterminer si c'est parce que la permission n'a pas été donnée, ou si c'est à cause d'un deny explicit.

Contrôle d'accès pour le contrôle GetEffectiveRights

Exemples

Supposons que l'on a un DIT avec les entrées et contrôles d'accès suivant :

```
o=sun.com
objectclass: top
objectclass: organization
o: sun.com
subtreeACI: grant:rsc#[all]#authnLevel:none:public:
subtreeACI: deny:rsc#[userPassword,subtreeACI,entryACI,salary]#authnLevel:none:public:
subtreeACI: grant:bvt#[entry]#authnLevel:none:public:
subtreeACI: grant:g#[entry]#authnLevel:limited:this:
subtreeACI: grant:worsc#[all]#authnLevel:limited:this:
subtreeACI: deny: wo[entryACI, subtreeACI, salary]#this
subtreeACI: grant:rscmo,w#[all]#authnLevel:strong:group:cn=adminGroup,ou=Groups,o=sun.com
subtreeACI: grant:bvtugeinad#[entry]#authnLevel:stronggroup: cn=adminGroup,ou=Groups,o=sun.com

cn=admin,o=sun.com
objectclass: top
objectclass: person
cn: admin
sn: admin
userPassword: secret
salary: 10000

ou=Groups,o=sun.com
objectclass: top
objectclass: organizationalUnit
ou: Groups

cn=adminGroup,ou=Groups,o=sun.com
objectclass: top
objectclass: groupOfUniqueNames
uniquemember: cn=admin,o=sun.com

ou=Eng,o=sun.com
objectclass: top
objectclass: organizationalUnit
ou: Eng

cn=Joe Engineer,ou=Eng,o=sun.com
```

```
objectclass: top
objectclass: person
cn: Joe Engineer
sn: Engineer
userPassword: secret
salary: 10000

ou=Sales,o=sun.com
objectclass: top
objectclass: organizationalUnit

cn=Joe Sales,ou=Sales,o=sun.com
objectclass: top
objectclass: person
cn: Joe Sales
sn: Sales
userPassword: secret
salary: 100000000000
```

La stratégie de contrôle d'accès dans ce DIT peut être décrite comme suit :

1. Les utilisateurs anonymes peuvent lire, rechercher et comparer les droits dans tout de DIT, excepté pour les attributs userPassword,subtreeACI,entryACI, et salary.
2. Tout utilisateur authentifié avec un mécanisme limité peut modifier les attributs de son entrée, excepté subtreeACI, entryACI, et salary.
3. Tous les utilisateurs peuvent lire tous les attributs dans son entrée.
4. Tout utilisateur authentifié avec un mécanisme limité peut récupérer les droits effectifs de sa propre entrée.
5. Les utilisateurs, authentifiés avec un mécanisme fort, dans le groupe cn=adminGroup,ou=Groups,o=sun.com peuent récupérer les droits effectifs dans tout de DIT.

Quelques exemples de requêtes pour obtenir les droits effectifs et les réponses :

Exemple 1 : Supposons que l'on fait une demande, authentifié au niveau strong, en tant que cn=admin,o=sun.com, avec comme base o=sun.com, un filtre "objectclass=*", et les attributs demandés "* entryACI", avec le contrôle getEffectiveRights et le sujet est cn=Joe Sales,ou=Sales,o=sun.com, et le MachineSubject spécifiant l'ipAddress et dnsName de la machine client et le authnLevel à limité.

Le résultat de recherche et les droits effectifs que l'on verra sont :

```
o=sun.com
objectclass: top
objectclass: organization
o: sun.com

entryLevelRights: bvt
attributeLevelRights: objectclass,o:rsc,entryACI:none
--
cn=admin,o=sun.com
objectclass: top
objectclass: person
cn: admin
sn: admin
userPassword: secret
salary: 10000
entryLevelRights: bvt
attributeLevelRights: objectclass,cn,sn:rsc,userPassword,salary,entryACI:none
--
ou=Groups,o=sun.com
objectclass: top
```

```
objectclass: organizationalUnit
ou: Groups
entryLevelRights: bvt
attributeLevelRights: objectclass,ou:rsc,entryACI:none
--
ou=Eng,o=sun.com
objectclass: top
objectclass: organizationalUnit
entryLevelRights: bvt
attributeLevelRights: objectclass,ou:rsc,entryACI:none
--
cn=Joe Engineer,ou=Eng,o=sun.com
objectclass: person
cn: Joe Engineer
sn: Engineer
userPassword: secret
salary: 10000
entryLevelRights: bvt
attributeLevelRights: objectclass,cn,sn:rsc,userPassword,salary,entryACI:none
--
ou=Sales,o=sun.com
objectclass: top
objectclass: organizationalUnit
entryLevelRights: bvt
attributeLevelRights: objectclass,ou:rsc,entryACI:none
--
cn=Joe Sales,ou=Sales,o=sun.com
objectclass: person
cn: Joe Sales
sn: Sales
userPassword: secret
salary: 100000000000
entryLevelRights: bvtg
attributeLevelRights: objectclass,cn,sn,userPassword:rscow,salary,entryACI:rsc
```

Interaction Client-Serveur

Le contrôle GetEffectiveRights demande les droits effectifs pour l'entrée/attribut demandé basé sur le sujet spécifié. Il y a 6 scénarios possibles qui peuvent se produire en résultat de ce contrôle :

1. Si le serveur ne supporte pas ce contrôle et que le client a spécifié la criticité, le serveur doit retourner unavailableCriticalExtension.
2. Si le serveur ne supporte pas ce contrôle et que le client n'a pas spécifié la criticité, le serveur doit ignorer le processus et traiter la recherche normalement.
3. Si le serveur supporte ce contrôle mais que pour une quelconque raison il ne peut le traiter et que le client a spécifié la criticité, le serveur devrait retourner unavailableCriticalExtension.
4. Si le serveur supporte ce contrôle mais que pour une quelconque raison il ne peut le traiter et que le client n'a pas spécifié la criticité, le serveur devrait retourner "no rights returned" et inclure "Unavailable" dans le contrôle GetEffectiveRightsResponse.
5. Si le serveur supporte ce contrôle et peut retourner les droits, il devrait inclure le contrôle GetEffectiveRightsResponse dans le message searchResult avec un code de résultat success.
6. Si la recherche échoue pour une quelconque raison, alors le serveur devrait omettre le contrôle GetEffectiveRightsResponse dans le searchResult.

L'application cliente est assurée que les droits correct sont retournés pour le scope de l'opération de recherche si et seulement si le contrôle GetEffectiveRightsResponse retourne les droits. Si le serveur omet ce contrôle, le client devrait assumer que le contrôle a été ignoré par le serveur.

Le contrôle GetEffectiveRightsResponse, si inclus par le serveur dans le searchResponse, devrait avoir le GetEffectiveRightsResult mis à "success" si les droits sont retournés ou à un code d'erreur approprié. Le serveur peut ne pas être en mesure de retourner un droits parce qu'il peut ne pas exister ; dans ce cas, la demande de droits est ignoré avec succès.