Access Control Model for LDAPv3

<draft-ietf-ldapext-acl-model-08.txt>

STATUS OF THIS MEMO

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Comments and suggestions on this document are encouraged. Comments on this document should be sent to the LDAP working group discussion list:

ietf-ldapext@netscape.com

COPYRIGHT NOTICE

Copyright (C) The Internet Society (2000). All Rights Reserved.

ABSTRACT

This document describes the access control model for the Lightweight Directory Application Protocol V3 (LDAPv3) directory service. It includes a description of the model, the schema for the model, and the LDAP control to the LDAP protocol. The current LDAP APIs are sufficient for the access control operations. A separate requirements document for access control exists [REQTS]. The access control model used the
requirements document as a guideline for the development of this specification and are reflected in this specification to the extent that the working group could agree on an access control model.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" used in this document are to be interpreted as described in [Bradner97].

1. Introduction

The ability to securely access (replicate and distribute) directory information throughout the network is necessary for successful deployment. LDAP’s acceptance as an access protocol for directory information is driving the need to provide an access control model definition for LDAP directory content among servers within an enterprise and the Internet. Currently LDAP does not define an access control model, but one is needed to ensure consistent secure access, replication, and management across heterogeneous LDAP implementations. The major objective is to provide a simple, usable, and implementable, but secure and efficient access control model for LDAP accessible directory content while also providing the appropriate flexibility to meet the needs of both the Internet and enterprise environments and policies. This document defines the model, the schema, and the LDAP control.

This model does not (and cannot) fully specify the behavior of the Access Control Model in a distributed environment (e.g. propagating access control information across servers and ACI administration) because there is no LDAP standard defining how to distribute directory data between LDAP servers. The behavior of the Access Control Model in distributed environments is beyond the scope of this model.

The following topics are deemed interesting and useful for future work, but are also beyond the scope of this model:

- Permissions based on object class
- Application permissions
- How to get initial entryACI and subtreeACI attributes in the directory is server specific
- Use of prescriptive ACIs and scoping via use of a ldapACISubEntry
- Required permissions for LDAP extended operations and LDAP controls, such as a proxy permission and permission extensibility
2. The LDAPv3 Access Control Model

Access Control mechanisms evaluate requests for access to protected resources and make decisions about whether those requests should be granted or denied. In order to make a grant/deny decision about a request for access to a protected resource, an access control mechanism needs to evaluate policy data. This policy data describes security-relevant characteristics of the requesting subject and the rules which govern the use of the target object.

No mechanism is defined in this document for storage of access control information at the server beyond indicating that the attribute holding access control information is an operational attribute.

The access control model defines

- What flows on the wire for interoperability

The existing LDAP protocol flows for ldap operations are used to manipulate access control information. These same flows on the wire apply when ACI is transmitted during replication. A set of permissions and their semantics with respect to ldap operations is defined. The permissions parallel the defined set of ldap operations. Encoding of access control information on the wire is per the LDAPv3 specifications.

There is a LDAP control defined, GetEffectiveRights. LDAP clients use the control to manage and administer access control policy enforced by LDAP servers.

Servers may store access control information in any way they choose. In particular, servers may use the access control mechanisms of their datastores to store and enforce LDAP access control, or they may implement/exploit access control managers external to their datastores. Datastores and external access control managers MAY implement any access control rule syntax and semantics they choose, but the semantics MUST be compatible with those defined in the section titled "Required Permissions for Each LDAP Operation".
Attributes and classes for application portability of access control information. Portable (LDAP) applications should only use the ACI in this model.

Two access control information attributes (entryACI and subtreeACI) for application portability. These attributes are used as input to the LDAP APIs so access control information can be addressed uniformly independent of how that information is accessed and stored at the server. These same attributes appear in LDIF output for interchange of access control information.

An access control information subentry class (ldapACISubEntry) and a set of attributes (supportedAccessControlSchemes which is used in the rootDSE, and accessControlSchemes which is used in the subentry ldapACISubEntry) to identify the access control mechanisms supported by a server and in a given part of the namespace, respectively.

A mechanism to control access to access control information: The access control information operational attributes, entryACI and subtreeACI, are also used to control access to access control information (controls access to itself). How to get initial entryACI and subtreeACI attributes in the directory is server specific and beyond the scope of this model.

Servers can support multiple access control mechanisms, but MUST be capable of supporting the LDAP Mechanism in the DIT scoped by the rootDSE (entire server’s DIT) for that server and SHOULD be capable of supporting the LDAP mechanism in an arbitrary part (subtree) of the DIT.

The accessControlSchemes attribute in the ldapACISubEntry indicates which access control mechanisms are in effect for the scope of that ldapACISubEntry. The supportedAccessControlSchemes attribute in the rootDSE indicates which access control mechanisms are supported by the server; those mechanisms are in effect in that server’s DIT unless overridden by a mechanism defined in a ldapACISubEntry elsewhere in that DIT.

Changing the value(s) of either the supportedAccessControlSchemes or accessControlSchemes attributes changes the mechanism(s) in effect for the scope of those attributes (where scope is either that of the rootDSE or ldapACISubEntry).

Through the use of the supportedAccessControlSchemes attribute in the rootDSE and the accessControlSchemes attribute in the ldapACISubEntry, it is possible to run multiple mechanisms in either the same subtree or separate subtrees. This might occur if the underlying datastore for the directory was accessible via LDAP and other applications. In this case, LDAP access should always be subjects to the LDAP access controls described in this document.
3. Access Control Mechanism Attributes

Two attributes are defined to identify which access control mechanisms are supported by a given server and by a given subtree: supportedAccessControlSchemes and accessControlSchemes. (We chose these names based on the X.500 attribute, AccessControlScheme which is single-valued and defined in X.501).

3.1 Root DSE Attribute for Access Control Mechanism

The server advertises which access control mechanisms it supports by inclusion of the ‘supportedAccessControlSchemes’ attribute in the root DSE. This attribute is a list of OIDs, each of which identify an access control mechanism supported by the server. By default, these are also the mechanisms in effect in subtrees beneath the root in that server unless overridden by a ldapACISubEntry (see section "Subentry Class Access Control Mechanism").

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

The access control mechanism defined is:
LDAPv3  <OID to be assigned>

Other vendor access control mechanisms MAY be defined (by OID) and are the responsibility of those vendors to provide the definition and OID.

3.2 Subentry Class Access Control Mechanism

A given naming context MUST provide information about which access control mechanisms are in effect for that portion of the namespace. This information is contained in a subentry (ldapACISubEntry class), derived from [SUBENTRY]. ldapACISubEntry MAY be used to define the scope of an access control mechanism. The value(s) held in the rootDSE attribute, supportedAccessControlSchemes, are the mechanisms in effect in subtrees beneath the root in that server unless overridden in a ldapACISubEntry further down the tree held by that server. The scope of that ldapACISubEntry is to the end of the subtree held by that server or until another ldapACISubEntry is encountered in that subtree held by that server. The ldapACISubEntry class is defined as:

{ <OID to be assigned>
The accessControlSchemes attribute MUST be in each ldapACISubEntry entry associated with a naming context whose access control mechanism is different from adjacent naming contexts supported by that directory server. accessControlSchemes lists the values (list of OIDs) that define the access control mechanisms in effect for the scope of that ldap access control subentry (either until another ldapACISubEntry is encountered in that subtree or end of subtree is reached on the server). Although, in general, this attribute will define only a single mechanism (single value), more than one mechanism MAY be in effect for the scope of that subtree.

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

4. The Access Control Information Attributes, Syntax, and Rules

The access control information syntax and attributes, entryACI and subtreeACI, are defined as:

(<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
)
Section 4.1 defines the ABNF and ASN.1 for these attributes, entryACI and subtreeACI.

The intent of the attribute definitions is to define a common interchange format and have a separation of responsibilities to allow different people to administer the different attribute types. (X.500 overcomes this by allowing access control on a per-value basis, which is complex). Any given LDAP server should be able to translate the defined attribute into meaningful requests. Each server should be able to understand the attribute; there should not be any ambiguity into what any part of the syntax means.

While the end goal is to have a common behavior model between different LDAP server implementations, the attribute definitions alone will not ensure identical ACL processing behavior between servers. Applicability and precedence rules for making access decisions are defined later in this section. The semantics of how a server interprets the ACI syntax are defined in the "Required Permissions for Each LDAP Operation" section of this document. Additionally, while the server must recognize and act on the attribute when received over the wire, there are no requirements for the server to physically store these attributes in this form.

The attribute definitions maintain an assumption that the receiving server supports ACI inheritance within the security model. If the server does not support inheritance, the receiving server must expand any inherited information based on the scope flag.

The attributes are defined so access control information (ACI) can be addressed in a server in an implementation independent manner. These attributes are used in typical LDAP APIs, in LDIF output of ACI and in transfer of ACI during replication. These attributes may be queried or set on all directory objects. The ABNF and definitions are given below.

4.1 The ABNF and ASN.1

4.1.1 ACI UTF-8 String Representation

The LDAP string encoding of values of the ACI syntax (<OID-aci-syntax>) is described by the following ABNF [ABNF]. This string representation MUST be supported.

    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

; permissions r, w, o, s, p, c, m work on attributes
; permissions a, d, e, i, n, b, v, t, u, g work on entries
; permissions for attributes and permissions for entries are
; never found in a single ACI

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

attribute = AttributeDescription
  ; The <AttributeDescription> rule is defined
  ; in Section 4.1.5 of [LDAPv3]

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"

; The <authzId> rule is defined in [AuthMeth]. It is repeated below for convenience.

authzId = dnAuthzId / uAuthzId

; distinguished-name-based authz id.
dnAuthzId = "dn:" dn
dn = utf8string ; with syntax defined in [UTF]

; 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 ]

Note that the colon following the "public" and "this" subject options exist only to simplify string parsing.

If an ACI is attempted to be added with an authz that is not understood,
then the server MUST reject that entryACI or subEntryACI value. This clarifies the statement in [AuthMeth] that allows for authzid may be expanded in the future.

See section titled ‘ACI Examples’ for examples of the string representation.

4.1.2 ACI Binary Representation

The ASN.1 type ACI is used to represent this syntax when transferred in binary form. This binary form SHOULD be supported.

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
}

-- An X.500 representation for an LDAP Attribute Description --
AttributeTypeAndOptions ::= SEQUENCE {
   type        AttributeType,
   type-name   UTF8String OPTIONAL,
   -- A hint of what LDAP textual name to use when encoding an
   -- AttributeTypeAndOptions as an <AttributeDescription>.
   options     SEQUENCE SIZE (1..MAX) OF CONTEXT.&Assertion OPTIONAL
   -- A future revision will constrain CONTEXT.&Assertion to be
   -- the context assertion syntax of the CONTEXT information
-- object defined by the X.500 working group to represent
-- LDAP attribute options in the X.500 protocols.
-- This is likely to be the UTF8String type.

GeneralSubject ::= CHOICE {
    anySubject          NULL,
    machineSubject  [1] MachineSubject,
    idBasedSubject      [2] IDBasedSubject
    -- may be expanded per [AuthMeth] --
}

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

IPAddress ::= UTF8String

-- The character contents of an IPAddress string are encoded
-- according to the <ipAddress> rule in Section 4.1.1.

DomainName ::= UTF8String

-- The character contents of a DomainName string are encoded
-- according to the <partialdomainname> rule in Section 4.1.1.

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

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 -- })
}

-- permissions read, write, obliterate, search,
--   searchPresence, compare, make work on attributes
-- permissions add, delete, export, import, renameDN,
-- browseDN, viewEntry, returnDN, unveil,
-- getEffectiveRights work on entries

END

4.2 The Components of entryACI and subtreeACI Attributes

This section defines components that comprise the access control information attributes, entryACI and subtreeACI.

The access control information in the entryACI attribute applies only to the entry in which it is contained.

The access control information in the subtreeACI attribute applies to each entry down the subtree unless it is overridden by an entry-specific entryACI whose values are more specific or another subtreeACI lower in the tree.

Use of prescriptive ACIs and scoping via use of a ldapACISubEntry is outside the scope of this document.

4.2.1 Access Rights and Permissions

Access rights can apply to an entire object or to attributes of the object. Access can be granted or denied. Either or both of the actions "grant" | "deny" may be used when creating or updating entryACI and subtreeACI.

Each of the LDAP access permissions are discrete. One permission does
not imply another permission. The permissions which apply to attributes and the entry parallel the type of ldap operations that can be performed.

Permissions which apply to attributes:

- r  Read            Read attribute values
- w  Write           Modify-add values
- o  Obliterate      Modify-delete values
- s  Search          Search entries with specified attributes
- p  searchPresence  Presence only filters
- c  Compare         Compare attributes
- m  Make            Make attributes on a new entry below this entry

1. r  Read

   If granted, permits attributes and values to be returned in a read or search operation.

2. w  Write

   If granted, permits attributes and values to be added in a modify-add operation.

3. o  Obliterate

   If granted, permits attributes and values to be deleted in a modify-delete operation.

4. s  Search

   If granted, permits attributes and values to be included in the search filter of a search operation.

5. c  Compare

   If granted, permits attributes and value to be used in a compare operation.

6. p  SearchPresence

   If granted permits attributes and values to be included in presence tests in a search filter.

7. m  Make

   The attribute permission "m" is required for all attributes that are placed on an object when it is created. Just as the "w" and
"o" permissions are used in the Modify operation, the "m" permission is used in the Add operation. Additionally, note that "w" and "o" have no bearing on the Add operation and "m" has no bearing on the Modify operation. Since a new object does not yet exist, the "a" and "m" permissions needed to create it must be granted on the new object’s parent. This differs from "w" and "o" which must be granted on the object being modified. The "m" permission is distinct and separate from the "w" and "o" permissions so that there is no conflict between the permissions needed to add new children to an entry and the permissions needed to modify existing children of the same entry.

Note: Modify-replace values of an attribute require both "w" and "o" permission.

Permissions that apply to an entire entry:

1. a  Add
   If granted, permits creation of an entry in the DIT subject to access control on all attributes and values to be placed in the new entry at time of creation. In order to add an entry, permission must also be granted to add all attributes that exist in the add operation.

2. d  Delete
   If granted, permits the entry to be removed from the DIT regardless of controls on attributes within the entry. Delete only works on leaf entries (same as X.500).

3. e  Export
   If granted, permits an entry and its subordinates (if any) to be exported; that is, removed from the current location and placed in
a new location subject to the granting of suitable permission at the destination. If the last RDN is changed, Rename is also required at the current location. In order to export an entry or its subordinates, there are no prerequisite permissions to contained attributes, including the RDN attributes; this is true even when the operation causes new attribute values to be added or removed as the result of the changes of RDN.

4. i Import

If granted, permits an entry and its subordinates (if any) to be imported; that is, removed from some other location and placed below the location to which the permission applies subject to the granting of suitable permissions at and below the source location. In order to import an entry or its subordinates, there are no prerequisite permissions to contained attributes, including the RDN attributes; this is true even when the operation causes new attribute values to be added or removed as the result of the changes of RDN.

5. n RenameDN

Granting Rename is necessary for an entry to be renamed with a new RDN. There are many cases here surrounding the use of this permission. See the section on 'Modify DN Operation'.

6. b BrowseDN

If granted, permits entries to be accessed using directory operations which do not explicitly provide the name of the entry.

7. v ViewEntry

If granted, permits entries to be accessed. This entry level read permission is useful as it is not dependent on the scope or aliasing properties of the entry.

8. t ReturnDN

If granted, allows the distinguished name of the entry to be disclosed in the operation result.

9. u Unveil

If granted, allows the presence of the entry to be disclosed in the case where access is denied to the entry according to the access control rules.

10. g getEffectiveRights

If granted, allows the effective rights on that entry and the
attributes within that entry to be returned, for _any_ subject.

### 4.2.2 Attributes

Attribute describes an attribute name in the form of a dotted decimal OID for that <attr>. If the string (OID) refers to an attribute not defined in the given server’s schema, the server SHOULD report an error. The use of "[entry]" or "[all]" helps to focus the permissions to either entry or attribute quickly, and hence helps in parsing.

"[entry]" means the permissions apply to the entire object. This could mean actions such as delete the object, or add a child object. When used in subtreeACI, the specified permissions (on the entry set of permissions are legal here - see the ABNF) apply to each entry in the subtree.

"[all]" means the permission set applies to all attributes of the entry. When used in subtreeACI, "[all]" applies to all attributes of each entry in the subtree.

If the keyword "[all]" and another attribute are both specified within an ACI, the more specific permission set for the attribute overrides the less specific permission set for "[all]."

### 4.2.3 Subjects and Associated Authentication

The following subjects are defined and MUST be supported:

- authzId, defined per [AuthMeth]
- group, defined as the distinguished name of a groupOfNames or groupOfUniqueNames entry
- role, asserts a subject’s organizational position and entitlement to perform the operations appropriate to that organizational position. It is implementation defined how the association between authorization id and the role dn is made.
- subtree, defined as some distinguished name of a non-leaf node in the DIT
- ipAddress, in IPv6 text format [IPV6]
- dnsName, a domain name or a wildcarded (left-most name or most specific part) domain name (see ABNF)
- public, defined as public access
- this, defined as the user whose name matches that of the entry being accessed

Other parties MAY define other subjects. It is the responsibility of those parties to provide the definition. Adding new subjects may inhibit interoperability.

A subject may be qualified by the level of authentication required for access to a given attribute(s) or entry. authnLevel defines the minimum requestor authentication level required for a given ACI. For a requestor's authentication level to exceed a minimum requirement, the requestor's level must meet or exceed that specified in the ACI.

The authentication levels defined, in increasing order of authentication, are:

- none - no name and no password, or name but no password
- weak - authentication mechanisms that are prone to both passive and active attacks \[ATTACK\]; for example, simple authentication (name and password)
- limited - authentication mechanisms that protect against passive attacks but may be prone to active attacks; for example, DIGEST-MD5
- strong - authentication mechanisms that protect against both passive and active attacks; for example, Kerberos with per-authentication or PKI authentication

The authnLevel applies to a subject as follows:

- If the ACI is a grant, then the authnLevel applies if the subject is authenticated at or above that level.
- If the ACI is a deny, then the authnLevel applies to that subject if authenticated at that level AND to all other subjects authenticated with levels below that.

Authentication level is always specified in an ACI. For grant, this means that you are granted access if you can prove your authentication via a strong authentication mechanism, such as a digital signature. For deny, the meaning of this is "you are denied access if you are Mr. X and you can prove it with a digital signature". If you are not Mr. X you are not denied access only if you can prove it (authenticate yourself) with a digital signature. In other words, everyone who does not authenticate with a digital signature is denied access. Everyone who authenticates with a digital signature is allowed access except Mr. X.

A recommended categorization of authentication mechanisms per authentication level may be offered separately. For each mechanism categorized in the recommendations, implementations SHOULD NOT assign a
higher authentication level, but MAY assign a lower authentication level. For mechanisms not covered by the recommendation, the implementation SHOULD specify a conservative level. Implementations SHOULD provide a means for the directory administrator to disable and/or lower the authentication level associated with a mechanism.

Two interesting subjects not explicitly included, but can be composed using subject and authnLevel are anonymous and authenticated.

- anonymous: authnLevel=none + anySubject(public)
- authenticated: authnLevel=weak + anySubject(public)

4.3 Access Control Decision Making

The ultimate goal of the Access Control Model is to define the way in which an LDAP server determines an access control decision for a given requested LDAP operation. In fact, a requestor may require several individual permissions in order to be authorized to carry out an LDAP operation and we define these in section 5. In this section we introduce the concepts and algorithm that allow the server to decide if a requestor has an individual permission on an individual resource. The concepts we require are firstly, the parameters which must be provided in order for the Access Control Decision Algorithm to determine whether the access is allowed or not. Secondly, we define precisely when a given ACI value will be involved in a given access control decision. Thirdly, this model defines some precedence rules which the Algorithm will use when dealing with more than one ACI value. Finally, we can define the Access Control Decision Algorithm which will determine the access decision. Throughout, we use the ABNF, when we need to refer to portions of individual ACI values.

4.3.1 The Parameters to the Access Decision Algorithm

The decision algorithm answers the question "Does the specified requestor have the specified permission on the specified resource?" The resource may be an entry (as for the Delete operation) or it may be an attribute within an entry (as for the Modify operation) so we characterize the resource like this: (targetEntry, targetAttribute OPTIONAL).

The requestor is identified primarily by his authorization ID (which may be omitted if the requestor has bound anonymously), but also includes other context information about the requestor so it is represented like this: (authzId OPTIONAL, ipAddress, dnsName, authnLevel).

The permission is one of the valid permissions defined by the model.

So, the parameters to the Access Control Decision Algorithm, which we will refer to collectively as "the decision parameters" are:
- resource: (targetEntry, targetAttribute OPTIONAL)
- permission: permissionParameter
- requestorSubject: (authzId OPTIONAL, ipAddress, dnsName, authnLevel)

If permissionParameter is an attribute-level parameter then targetAttribute must be specified; if it is an entry-level permission then targetAttribute is ignored.

The output is either "Access Allowed" or "Access Denied".

4.3.2 Applicability Rules

The applicability rules define whether a given ACI value, or portions of it, apply to some given decision parameters.

4.3.2.1 Applicability Rules for Scope Types

These rules determine whether a specific ACI applies to the targetEntry part of the decision parameters.

- If the ACI in question is an entryACI, then ACI applies to the resource if the ACI is an attribute of the entry targetEntry.

- If the ACI in question is a subtreeACI, then ACI applies to the resource if targetEntry is part of the subtree defined by the entry containing the ACI.

4.3.2.2 Applicability Rules for Permissions

If permissionParameter is an entry-level permission, then the ACI in question applies if permissionParameter is mentioned in the permissions list of the ACI.

If permissionParameter is an attribute-level permission, then the ACI in question applies if permissionParameter is mentioned in the permissions list and the ACI’s attribute list applies to the target attribute per "Applicability Rules for Attributes".

Note that for an ACI which contains both grant and deny permissions lists, the grant permission list may not be available for the purposes of this applicability rule--see point 3 in the "Applicability Rules for Subjects".
4.3.2.3 Applicability Rules for Attributes

If an attribute is not specified as part of the resource, then this rule does not apply. If an attribute is specified, then the ACI in question applies if its attribute list is [all] or if targetAttribute is explicitly mentioned in the ACI’s attribute list.

In the case where targetAttribute is an attribute type with options (e.g. sn;lang-en;lang-uk), it is applicable if the ACI’s attribute list mentions a less specific form of targetAttribute. For example, if the list contained "sn;lang-en", then that list applies to the attribute "sn;lang-en;lang-uk", but not the attribute "sn".

4.3.2.4 Applicability Rules for Subjects

Call the subject portion of the ACI in question aciSubject. Then to determine if aciSubject applies to requestorSubject we apply the following rules:

1. The ACI in question is a grant ACI. Then the ACI applies if both the context and pureSubject portions of aciSubject apply, as defined in "Applicability Rules for Context" and "Applicability Rules for pureSubject" below.

2. The ACI in question is a deny ACI. There are three possibilities:
   a. The pureSubject part applies (according to "Applicability Rules for pureSubject"). Then the ACI applies to requestorSubject.
   b. The pureSubject part does not apply. Then the ACI applies to any requestorSubject with an authnLevel less than the authnLevel of the ACI.
   c. Otherwise the ACI does not apply.

3. The ACI in question is both a deny and grant ACI. There are three possibilities:
   a. aciSubject applies when evaluated as a grant ACI (per 1 above). Both the grant permissions and deny permissions of the ACI are available for the purpose of the "Applicability Rules for Attributes and Permissions".
   b. aciSubject does not apply as a grant ACI but does apply as a deny ACI (per 2 above). The deny permissions of the ACI are available for the purpose of the "Applicability Rules for Attributes" and the "Applicability Rules for Permissions".
   c. aciSubject does not apply when evaluated as either a grant ACI or a deny ACI. The ACI does not apply to
requestorSubject.

Note: the deny behavior with authnLevel deserves explication. In the case where an ACI denies access to a given subject with a given authnLevel, then naturally it will deny access to that given subject authenticated at authnLevel or above. Similarly, another user authenticated at authnLevel or above, to which the pureSubject part does not apply, will not be denied those rights by that ACI.

The interesting case is a user authenticated at a lower level than authnLevel. For such a user the ACM takes the view that as that user has not proved to the system, to a sufficient degree of confidence, that the pureSubject portion does not apply to him, then to be safe, he will be denied those rights.

So if you deny access to a particular authzId with authnLevel of "none", then that authzId will be denied access at any authentication level, but it will not affect any other requestors. On the other hand, if you deny access to a particular authzId with an authnLevel of "strong", then that will deny access to that user when authenticated strongly AND deny access to ANY users authenticated with lower authentication levels.

### 4.3.2.5 Applicability Rules for pureSubject

If aciSubject is of type anySubject, then it applies to requestorSubject.

If aciSubject is of type machineSubject, then if the ipAddress or dns name from requestorSubject matches the ipAddress or dns name range in aciSubject, then the ACI applies to requestorSubject if it is a deny ACI or the deny part of a grant/deny ACI. A grant ACI (or the grant part of a grant/deny ACI) never applies if the subject is ipAddress: or dns:.

The note at the end of the "Precedence of Subjects within a Scope" explains the reasoning behind this rule.

If the aciSubject is of type idBasedSubject, then it applies according to the definition in "Applicability Rules for idBasedSubject".

### 4.3.2.6 Applicability Rules for Context

The context of aciSubject applies to requestorSubject if authnLevel from requestorSubject is greater than or equal to the authnLevel specified in the context part of aciSubject.

### 4.3.2.7 Applicability Rules for idBasedSubject

If idBasedSubject is of type thisSubject, then it applies to requestorSubject if authzId from requestorSubject is of type "dn" and is equal to the DN of the resource.
If idBasedSubject is of type oneSubject, then it applies to requestorSubject if authzId from requestorSubject is equal to the authzId specified in aciSubject.

If idBasedSubject is of type setOfSubjects, then it applies to requestorSubject if authzId from requestorSubject is defined to be in the set specified in aciSubject (i.e. is in that group, role or subtree). The "Precedence of Subjects within a Scope" includes rules for determining membership in a setOfSubjects.

4.3.3 Precedence Rules

The precedence rules allow the Access Control Decision Algorithm to determine which ACI values should take precedence over others.

4.3.3.1 Precedence of Scope Types

1. Entry
2. Subtree

4.3.3.2 Precedence of Position in the DIT

For a given subject DN (including authentication level) and target DN, subtreeACI lower in the tree take precedence over those higher in the tree.

4.3.3.3 Precedence of Subjects within a Scope

1. ipAddress or dns in a deny ACI or the deny part of a grant/deny ACI
2. authzId distinguished name ("authzId-dn:"), or authzId userid ("authzId-u:"),
3. this
4. role
   
   If the DN of a role or a group appears in a role (e.g. appears as a value of roleOccupant in an organizationalRole), it is recursively expanded. For determination of precedence, the resulting expanded collection of names is considered to have come from a role regardless of the original source.
5. group

If the DN of a role or a group appears in a group, it is recursively expanded. For determination of precedence, the resulting expanded collection of names is considered to have come from a group regardless of the original source.

6. subtree

Subtrees may contain groups and/or roles. They should be recursively expanded. For determination of precedence, members of groups or occupants of roles that apply because (after recursive expansion) the group or role is contained in a subtree are considered to have come from the subtree regardless of the original source.

7. public

The precedence of ipAddress and DNS names are treated specially, and depend on the type of ACI involved. Typical situations are:

- If an ACL says to grant on the basis of IP address but deny on the basis of some other attribute (username, group, etc....), then the behavior we want is "deny". Rationale: the user is prohibited from access, regardless of where he’s sitting.

- If an ACL says to deny on the basis of IP address but grant on the basis of some other attribute (username, group, etc....), then the behavior we want is also "deny". Rationale: the user is allowed access, but not from where he’s sitting.

In addition, a grant to an ipAddress with no other applicable ACI is very dangerous from a security point of view, since it would grant permissions to ANY user who has access to the machine with the given address. Thus ipAddress and dns subjects can be used only to deny permission, not to grant them. The "Applicability Rules for pureSubject" enforce this.

4.3.3.4 Precedence of Attribute Specifications

Access controls specifying "[all]" attributes are of lower precedence than specific lists.

4.3.3.5 Precedence of grant/deny

Deny takes precedence over grant.
4.3.3.6 Default

Deny is the default when there is no access control information or when evaluation of the access control information yields no result that allows the requester access.

4.3.4 Access Control Decision Algorithm

The Access Control Decision Algorithm takes as input the decision parameters defined above and produces a grant or a deny decision.

In the case where we are interested in the permission set for a set of entries and attributes (as in the case of evaluating the effective rights in section 9), then we must apply the algorithm for each entry/attribute and permission pair we are interested in. Naturally implementers are free to implement any algorithm which produces the same decision given the same input and ACI values in a DIT.

The algorithm has two phases. First, all the potentially applicable ACI values are sorted into an ordered list of sets of ACI values of the same precedence. Then this list is scanned in order to find the set of ACIs which will determine the access decision.

Phase 1: Order the potentially applicable ACI values

1. Determine all the entryACI and subtreeACI values that apply to targetEntry, according to the "Applicability Rules for Scope Types".

2. Sort these ACIs into a list of sets of ACIs of equal precedence, according to the "Precedence of Scope Types" and "Precedence of Position in the DIT" rules.

3. Determine which of the collected ACI values from step 1 apply to requestorSubject using the "Applicability Rules for Subjects". All the ACI values which do not apply to this subject are discarded.

4. The list of sets is sorted according to subject type from the "Precedence of Subjects within a Scope" rules.

5. Now we split the list into two separate lists keeping the same relative ordering of sets--one list has the sets that just contain ACI values that refer to entry-level permissions and the other has the sets that just contain ACI values that refer to attribute-level permissions.

6. Each set in the attribute-level list of sets is further divided into a list of sets of equal precedence according to "Precedence of Attributes Specification".
Note: At this point we have two lists of sets of ACI values, one dealing with entry-level permissions the other dealing with attribute-level permissions. The lists have been sorted according to Scope, Position, Subject and (for the attribute-level list) Attribute Specification precedence rules.

Phase 2: Scan the lists to determine the access decision:

1. If permissionParameter is an entry-level permission (so that the optional targetAttribute field is not specified), then scan the list of entry-level sets in order. The first set in the list that contains an ACI that applies to permissionParameter (as defined in the "Applicability Rules for Permissions" rules) determines the access decision—if an ACI in the set grants permissionParameter and no other denies it, then access is granted, otherwise access is denied.

2. If permissionParameter is an attribute-level permission then scan the list of attribute-level sets in order. The first set in the list that contains an ACI that applies to permissionParameter AND applies to targetAttribute (as defined in the "Applicability Rules for Attributes" and "Applicability Rules for Permissions") determines the access decision—if an ACI in the set grants permissionParameter and no other denies it, then access is granted, otherwise access is denied.

4.3.5 Precedence/Inheritance Access Decision Example
The examples in this section refer to the following directory tree:

```
dc=com
  +-----------------+-----------------+
  |                |                |
  dc=tivoli       dc=sun
  |                |                |
  cn=ellen        cn=rob
```

The ACI at 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

The ACI at 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

The ACI at 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

Example #1

subject: ("cn=rob, dc=sun, dc=com", ipaddress, dns name, strong):
resource: ("cn=ellen, dc=tivoli, dc=com", salary)
permission: "w"

Phase 1: Find all applicable ACI based on the Applicability of Scope Types.

   {1, 2, 3, 4, 5, 6, 7, 8, 9}

Sort by Precedence of Scope Type and Precedence of Position in DIT:

   {8, 9}, {6, 7}, {1, 2, 3, 4, 5}

Determine which ACI are applicable based on the Subject:

   {6, 7}, {1, 2, 3, 4, 5}

Sort by Precedence of Subjects within Scope:

   {6, 7}, {4, 5}, {1, 2, 3}

Separate permissions applicable to entry and those applicable to attributes:

   Entry: {7}, {5}, {3}
   Attr:   {6}, {4}, {1, 2}

Sort the permissions applicable to attributes by precedence of attribute specification:

   Entry: {7}, {5}, {3}
   Attr:   {6}, {4}, {2}, {1}

Phase 2: Since "w" is an attribute permission, look at the Attr list. ACI 6 in the first sub-list mentions "w" and salary (via [all]) so this set defines the access--which is deny.
Example #2

subject: ("cn=rob,dc=sun,dc=com", ipaddress, dns name, limited):
resource: ("cn=ellen,dc=tivoli,dc=com", salary)
permission: "w"

Phase 1: First the ACIs are ordered into the following sets whose subject matches the subject tuple:

Entry: {7}, {3}
Attr:  {6}, {2}, {1}

Phase 2: For ACI 6 in the first set, which matched the subject because of the deny portion and limited < strong, the permissions available are "mow". So, this ACI applies to "w" and salary (via [all]) and the access is "deny".

Example #3

subject: ("cn=rob,dc=sun,dc=com", ipaddress, dns name, limited):
resource: ("cn=ellen,dc=tivoli,dc=com", salary)
permission: "r"

Phase 1: First the ACIs are ordered into the following sets whose subject matches the subject tuple:

Entry: {7}, {3}
Attr:  {6}, {2}, {1}

Phase 2: As the grant portion of ACI 6 is not active, the first set that contains an ACI that applies to "r" and salary is {2}. As 2 denies access, access is denied.

Example #4

subject: ("cn=rob,dc=sun,dc=com", ipaddress, dns name, limited):
resource: ("cn=ellen,dc=tivoli,dc=com", cn)
permission: "r"

Phase 1: First the ACIs are ordered into the following sets whose subject matches the subject tuple:

Entry: {7}, {3}
Attr:  {6}, {2}, {1}

Phase 2: As the grant portion of ACI 6 is not active, the first set that contains an ACI that applies to "r" and cn is {1}. As 1 grants access, access is granted.
5. Required Permissions for each LDAP Operation

This section defines the required permissions for each LDAP operation. Even if these requirements are satisfied, the server may refuse to carry out the operation due to other implementation specific security considerations. For example, a server may refuse to modify an entry because the database where that entry resides is in read only mode. Another example might be that although LDAP access control has been specified on the userPassword attribute a server may refuse modifications due to some server specific policy governing access to passwords.

Here, we specify the rights required by a user when performing an LDAP operation in terms of the LDAP permissions specified above. Recall that "a, d, e, i, n, b, v, t, u" are permissions that apply to entries as a whole while permissions "r, s, p, w, o, c, m, g" apply to attributes within entries.

Required permissions for LDAP extended operations and LDAP controls SHOULD be defined along with their specifications. These requirements could be expressed in terms of this model, for example by requiring one of the existing permissions on some particular entry or attribute. This version of the LDAP access control model does not offer any mechanism to extend the permission set or aci syntax to accommodate extended operations or controls.

For the following, assume that the authorization identity of the user doing the operation is authzId.

5.1 Bind Operation

This model does not require any permissions to allow a bind operation to proceed.

5.2 Search Operation

Recall that the parameters of the Search operation per RFC 2251 [LDAPv3] Section 4.5 are:

\[
\text{SearchRequest ::= [APPLICATION 3] SEQUENCE {}
\text{baseObject LDAPDN,}
\text{scope ENUMERATED {}
\text{baseObject (0),}
\text{singleLevel (1),}
\text{wholeSubtree (2) },
\text{derefAliases ENUMERATED {}
\text{neverDerefAliases (0),}
\text{derefInSearching (1),}
\text{derefFindingBaseObj (2),}
\text{}}}
\]
Suppose a server is processing a search request from user authzId with parameters as above and is processing the entry with dn candidateDN to decide if it may be returned or not.

Then the permissions required by authzId that need to be evaluated are as follows:

1. permission "b" to the entry candidateDN if the entry is in the scope of the search but is not the base entry.

   If this permission is not granted then the dn candidateDN MUST NOT be returned nor any attribute type nor attribute value from this entry.

   Note: the "b" permission is included to allow different browsing or discovery rights to be assigned to different classes of users.

2. permission "v" to the entry candidateDN

   If this permission is not granted then the dn candidateDN MUST NOT be returned nor any attribute type nor attribute value from this entry.

   Note A: The "v" permission is the entry level read permission. This is included as it is useful to have one simple permission (not dependent on scope or aliasing) that protects all the components of an entry; the dn and the attributes.

   Note B: Disclosing the full distinguished name of an entry will inevitably reveal the names of its ancestors. This issue is discussed in detail below.

3. permission "p" or "s" to each attribute appearing in a search filter presence test during the evaluation of the search filter.

   permission "s" is required on attributes appearing in non-presence tests (see RFC2254, section 3: equalityMatch, substrings, greaterOrEqual, lessOrEqual, present, approxMatch, extensibleMatch) during the evaluation of the search filter.

   The above statement covers the case where the attributes are being evaluated as part of an extensibleMatch (RFC 2251 section 4.5.1) which appears in the filter. In the case where the dnAttributes field of the extensibleMatch is true (and so the filter test is...
applied to the naming attributes in the dn of the entry) then we do not require any access checks to the attributes of the dn as access to these is taken to be granted by the "v" permission, which has already been required above.

If there is an attribute in a filter element to which the required permission is not granted then that filter element evaluates to "Undefined" of the three-valued-logic of X.511(93).

Note: The motivation for the "p" permission is that if you have full filter rights on an attribute then in some cases that could be operationally the same as having read permission i.e. you could quickly determine the attribute value, and this may not be desirable. For example, if the type of the attribute was integer then with full filter permissions you could quickly determine the value by doing a sequence of binary chop style searches using ">" and "<". Whereas, with just the presence test ability, you would not have right to do those kind of searches, but you would be able to test for the presence of the attribute.

4. permission "r" to each attribute in the attribute list AttributeDescriptionList (or all attributes in the entry candidateDN if AttributeDescriptionList is ") whose type and/or value will be returned.

Note A: The presence of an attribute in an entry is only ever volunteered by the server if "r" permission is granted to it, though a user may infer the presence of an attribute with "s" or "p" permission by using a presence test on that attribute in the search filter.

Note B: Although both "r" and "s" permissions will typically be granted to attributes we keep both permissions as there are cases where the distinction is useful. A reverse telephone lookup is an example of granting "r" but not "s" permission.

5. permission "t" to the entry candidateDN

If this permission is not granted then the dn candidateDN MUST NOT be returned. If the server knows of an alias for the entry, this alias may be returned instead. If no alias name is available then the entry candidateDN MUST be omitted from the search results.

Note: Disclosing the full distinguished name of an entry will inevitably reveal the names of its ancestors. This issue is discussed in detail below.

6. Disclose on error for the Search operation

If there is no entry in the scope of the search which satisfies item 1 (browse right on the candidate entry) and item 2 (read
level permission on the entry) and item 3 (right to use the filter on that entry) then the "u" permission on the base entry must be evaluated. If "u" (disclose on error) is not granted to the base entry then the operation MUST fail with a "no such object error" and the matchedDN of the LDAPResult MUST be set to ".". If "u" is granted to the baseObject then the empty set of results is returned.

Note: the point here is that if you do not have the right to discover at least one entry in the scope of the search then the disclose on error mechanism is there to prevent you discovering the base entry of the search. The "t" permission is not considered here as it is not conceptually a permission involved in the discovery of entries but rather in how they are returned (dn vs. alias).

5.2.1 Protecting the naming attributes of DNs

Protecting the naming attributes in an entry’s dn presents a problem for access control. The issue is that if access is granted to the dn of a given entry, then via the naming attributes this dn contains, access is also also granted to attribute values in other entries. In addition, knowledge about the existence of ancestor entries of a given entry is also disclosed by the entry’s dn.

It could be argued that there should be consistency in the ability of a given requestor to see attribute values in the dn of an entry and his ability to see those attributes in the entry where they actually reside. Similarly, it could be argued that there should be consistency in the ability of a requestor to see an entry and his ability to see its ancestor entries.

The main reason we do not require such cross checking of permissions is because of the extra work it entails for the server. There is a trade off between the consistency this cross checking guarantees and the work it takes to do that cross checking.

For LDAP servers the trade off has been to go in favor of speed. In addition there are some other points which mitigate these kind of inconsistencies. Firstly, it appears to be difficult to produce a real world example where they really cause a problem. Secondly there are other methods of hiding DNs (and hence protecting the naming attribute and ancestor information they contain) which are outside the scope of access control, for example aliasing and LDAP proxying.

5.3 Modify Operation

Recall that the parameters of the Modify operation per [RFC2251] Section 4.6 are:
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 }

Then the permissions required by authzId that need to be evaluated are as follows:

1. permission "w" to each attribute being added to object

   If this permission is not granted to such an attribute, then the operation MUST fail.

2. permission "o" to each attribute for which a value is being deleted from object

   If this permission is not granted to such an attribute, then the operation MUST fail.

3. permissions "o" and "w" to each attribute being replaced in object

   If each of these items is satisfied then the operation is allowed by the access control model. If one of these items is not satisfied, then the operation MUST fail. In this case, if "u" (discloseOnError) is granted to object, then the usual error codes (such as noSuchObject, attributeOrValueExists, noSuchAttribute and insufficientAccess) and matchedDN value may be returned; if "u" is not granted to object then nosuchObject MUST be returned and matchedDN set to "".

5.4 Add Operation

Recall that the parameters of the Add operation per RFC2251 [LDAPv3] Section 4.7 are:

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

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

Then the permissions required by authzId that need to be evaluated are as follows:

1. permission "a" to the parent of entry

   The access rights required for the creation of a root entry in the Directory are beyond the scope of this document. They will be vendor specific.

2. permission "m" to the parent of entry for each attribute being added to entry

If each of these items is satisfied then the operation is allowed by the access control model. If one of these items is not satisfied, then the operation MUST fail. In this case, if "u" (discloseOnError) is granted to the parent of entry, then the usual error codes (such as noSuchObject, entryAlreadyExists, and insufficientAccess) and matchedDN value may be returned; if "u" is not granted to the parent of entry, then nosuchObject MUST be returned and matchedDN set to "".

Note A: We require "m" permission to each attribute to prevent an entry from acquiring "unintended" rights (via group or role membership), to stop a "rogue" ACI being added that would prevent even admins deleting the entry, and for general consistency with the MODIFY operation.

5.5 Delete Operation

Recall that the parameters of the Delete operation per RFC2251 [LDAPv3] Section 4.10 are:

   DelRequest ::= [APPLICATION 10] LDAPDN

Then the permissions required by authzId that need to be evaluated are as follows:

1. permission "d" to the entry in the Delete request

If each of these items is satisfied then the operation is allowed by the access control model. If one of these items is not satisfied, then the operation MUST fail. In this case, if "u" (discloseOnError) is granted to the entry to be deleted, then the usual error codes (such as noSuchObject, and insufficientAccess) and matchedDN value may be returned; if "u" is not granted to object then nosuchObject MUST be returned and matchedDN set to "".

Note: One could also require the "o" permission to be granted to allow
the operation to proceed, but customer experience has shown that the requirement of the additional permission is not useful nor expected, and X.500 requires only the "d" permission.

5.6 Modify DN Operation

Recall that the parameters of the Modify DN operation per RFC2251 [LDAPv3] Section 4.6 are:

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

The variants of the ModifyDN operation are listed below. Combinations of the write, obliterate, import, export and renameDN permissions are needed for each variant.

1. Rename an entry by moving the conceptual RDN flag between two existing attribute values, without altering any attribute values at all. Permissions needed are renameDN.

2. Rename an entry by adding a new attribute value. Permissions needed are renameDN and write.

3. Rename an entry using an existing attribute value and delete the current attribute value. Permissions needed are renameDN and obliterate.

4. Rename an entry adding a new attribute value and deleting the current attribute value. Permissions needed are renameDN, write, and obliterate.

5. Move an entry to a new place in the DIT, keeping its existing RDN as is. Permissions needed are import and export.

6. Move an entry to a new place coupled with 1. above Permissions needed are import, export, and renameDN.

7. Move coupled with 2. above. Permissions needed are import, export, renameDN, and write.

8. Move coupled with 3. above. Permissions needed are import, export, renameDN, and obliterate.

9. Move coupled with 4. above. Permissions needed are import, export, renameDN, write, and obliterate.

For a given case, if the required permissions are granted, then the
operation is allowed by the access control model. If, for a given case, the required permissions are not granted, then the operation MUST fail. If the access control failure is due to a missing attribute or entry permission on entry, then if "u" (discloseOnError) is granted to entry, then the usual error codes (such as noSuchObject, attributeOrValueExists and insufficientAccess) and matchedDN value may be returned; if "u" is not granted to entry then nosuchObject MUST be returned and matchedDN set to "". Similar logic applies if the access control failure was due to a missing permission on newSuperior.

5.7 Compare Operation

Recall that the parameters of the Compare operation per [LDAPv3] Section 4.10 are:

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

Then the permissions required by authzId that need to be evaluated are as follows:

1. permission "c" to the attribute in entry on which the comparison is being made.

If each of these items is satisfied then the operation is allowed by the access control model. If one of these items is not satisfied, then the operation MUST fail. In this case, if "u" (discloseOnError) is granted to entry, then the usual error codes (such as noSuchObject, and insufficientAccess) and matchedDN value may be returned; if "u" is not granted to entry then nosuchObject MUST be returned and matchedDN set to "".

5.8 Abandon Operation

Recall that the parameters of the Abandon operation per [LDAPv3] Section 4.6 are:

\[
\text{AbandonRequest ::= [APPLICATION 16] MessageID} \]

authzId always has the right to send an Abandon Operation for an operation he previously initiated.

5.9 Extended Operation

Recall that the parameters of the Extended operation per [LDAPv3] Section 4.12 are:
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
    requestName      [0] LDAPOID,
    requestValue     [1] OCTET STRING OPTIONAL }

The access required for an Extended Operation is beyond the scope of this document. The required access will normally be defined by the implementer of the extended request.

6. Required Permissions for Handling Aliases and References

Use of aliases and referrals are part of LDAPv3. However, neither is particularly well-defined. Alias objects and attributes are defined in RFC 2256 as derived from X.500, but LDAPv3 does not explicitly define their semantics or behavior. X.500 does define alias semantics and behavior with respect to access control; we define its behavior in LDAPv3 based on the X.511 (1993) [X511], section 7.11.1. Referrals and knowledge information are still under design in LDAPv3; they are defined in X.500, however, X.500 does not specify their semantics and behavior with respect to access control. We define their semantics and behavior in LDAPv3 in terms that should be independent of the future LDAPv3 definition of referrals and knowledge information.

6.1 ACI Distribution

Currently there is no LDAP standard defining how to distribute directory data between LDAP servers. Consequently this model cannot fully specify the behavior of the Access Control Model in a distributed environment. The case of distribution via referrals is treated in the "Referrals" section below. In the case of chaining (where one LDAP server forwards a request to another on behalf of a client) then it is server specific how the access control model behaves in this environment. Similarly it is server specific how the server determines whether the chaining of an operation is permitted in the first place. For example, the implementation may choose to regard the local naming context and the remote subordinate naming context as separate Access Control Specific Areas, or it may regard the DIT as one Access Control Specific Area and implement mechanisms to propagate access control information between the two servers. The behavior of the Access Control Model in distributed environments such as these is beyond the scope of this model.

6.2 Aliases

There are two things to protect with respect to aliases: the real name of the aliased object and the location of the server holding it.

If alias de-referencing is required in the process of locating a target entry, no specific permissions are necessary for alias de-referencing to
take place. Access control is enforced at the object pointed to by the alias.

If alias de-referencing would result in a continuationReference (e.g. from a search operation), then browse permission is required to the alias entry and read permission is required to the attribute. Requiring these permission closes the hole of discovery.

6.3 Referrals

If a referral is to be followed, no specific permissions are necessary for the ldap client to follow the referral. Access control is enforced at the referenced object. If a referral is returned, then browse is required on the entry and read permission is required to the attribute containing the referral (we cannot name this attribute exactly today because there are no RFCs on this). If the server implements a default referral, then no special permissions are required to read and return that referral. Requiring these permissions closes the hole of discovery. In the default case, it is assumed that a default referral is public.

7. Controlling Access to Access Control Information

The entryACI and subtreeACI attributes are used to specify control for who has permission to set/change access control information (entryACI and subtreeACI). The entryACI and subtreeACI attributes/OIDs are just another attribute described with set of rights and permissions, and subject as a value of the entryACI and subtreeACI attributes. (See the example in the "ACI Examples" section).

If the policy for controlling the entryACI and subtreeACI attributes are not specified for any object in the tree, behavior is implementation defined. For instance, if no object anywhere in the tree defines the access for entryACI/subtreeACI within the entryACI/subtreeACI attributes, then the server could simply assert that the ‘root DN’ is considered the policy owner (controller for controlling access control) for all objects.

8. ACI Examples

Note that in the examples, the form "OID.<attrname>" refers to the OID in dotted decimal form for the attribute <attrname>. This shorthand notation is used only for the examples. In implementation, the dotted decimal form of the OID is used.
8.1 Attribute Definition

The following examples show the access required to control access to the entryACI and subtreeACI attributes. The first example shows controlling the access control on an individual entry and its attributes. The second example shows controlling the access control on a subtree. The authnLevel is set so that a reasonably secure form of authentication is required, since changing ACI has significant security consequences.

entryACI: grant:rwo#
   OID.entryACI#authnLevel:limited:role:cn=aciAdmin

subtreeACI: grant:rwo#
   OID.subtreeACI#authnLevel:limited:role:cn=aciAdmin

The next example shows a subtreeACI attribute where a group "cn=Dept XYZ, c=US" is being given permissions to read, search, and compare attribute attr1. The permission applies to the entire subtree below the node containing this ACI.

subtreeACI: grant;rsc#
   OID.attr1#authnLevel:weak:group:cn=Dept XYZ,c=US

The next example shows an ACI attribute where a role "cn=SysAdmins,o=Company" is being given permissions to browseDN, viewEntry, and returnDN for objects below this node. The role is also being given permission to read, search, and compare to attribute attr2 and read, search, compare, write, obliterate to attr3. The permissions apply to the entire subtree below the node containing this ACI.

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

8.2 Modifying the entryACI and subtreeACI Values

Modify-Replace works as defined in the ldap operation modify. If the attribute value does not exist, create the value. If the attribute does exist, replace the value. If the entryACI/subtreeACI value is replaced, all entryACI/subtreeACI values are replaced. To modify the entryACI/subtreeACI attributes requires you to have 'w' and 'o' permission on the entryACI/subtreeACI attribute (recall that a value of entryACI/subtreeACI specifies entryACI/subtreeACI so one can control access to the entryACI/subtreeACI attribute). The examples in this section assume that you have access to control entryACI/subtreeACI.
A given subtreeACI for an entry:

    subtreeACI: deny:rw#[all]#authnLevel:weak:group:cn=Dept ABC
    subtreeACI: grant:r#OID.attr1#authnLevel:weak:group:cn=Dept XYZ

perform the following change:

    dn: cn=someEntry
    changetype: modify
    replace: subtreeACI
      subtreeACI: grant:rw#[all]#authnLevel:weak:group:cn=Dept LMN

The resulting ACI is:

    subtreeACI: grant:rw#[all]#authnLevel:weak:group:cn=Dept LMN

( subtreeACI values for Dept XYZ and ABC are lost through the replace )

During an ldapmodify-add, if the ACI does not exist, the create the ACI
with the specific entryACI/subtreeACI value(s). If the ACI does exist,
then add the specified values to the given entryACI/subtreeACI. For
example a given ACI:

    subtreeACI: grant:rw#[all]#authnLevel:weak:group:cn=Dept XYZ

with a modification:

    dn: cn=someEntry
    changetype: modify
    add: subtreeACI
      subtreeACI: grant:r#OID.attr1#authnLevel:weak:group:cn=Dept XYZ

would yield an multi-valued ACI of:

    subtreeACI: grant:rw#[all]#authnLevel:weak:group:cn=Dept XYZ
    subtreeACI: grant:r#OID.attr1#authnLevel:weak:group:cn=Dept XYZ

To delete a particular ACI value, use the regular ldapmodify - delete
syntax

Given an ACI of:

    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
would yield a remaining ACI on the server of

```
subtreeACI: grant:r#[all]#authnLevel:weak:group:cn=Dept XYZ
```

The attributes which are defined for access control interchange may be
used in all LDAP operations.

Within the ldapmodify-delete operation, the entire acl may be deleted by specifying

```
dn: cn = some Entry
changetype: modify
delete: entryACI
```
```
dn: cn = some Entry
changetype: modify
delete: subtreeACI
```

In this case, the entry would then inherit its ACI from other nodes in
the tree depending on the inheritance model.

Similarly, if all values of entryACI and subtreeACI are deleted, then
the access control information for that entry is defined by the
inheritance model.

### 8.3 Evaluation

These examples assume that the ACI entries listed in each example are
the only ACI which applies to the entry in question; if backing-store
ACI also exists, the effective policy may be different from that listed
in each example.

Assume cn=jsmith is a member of group cn=G1. Assume cn=jsmith is a
member of group cn=G2.

Example #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
```

What rights does cn=jsmith have to attr2 of o=XYZ,c=US? Read-write (rw)
access; ACI is combined because both subjects (group) have same
precedence.
Example #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

What rights does cn=jsmith have to attr3 of o=XYZ, c=US? Read access; write is denied (deny has precedence over grant).

Example #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

What rights does cn=jsmith have to o=XYZ, c=US? Make(m) on attributes attr5, cn, and sn and Add(a) on the entry. These are the minimal yet sufficient permissions to create a new object, cn=New, o=XYZ, c=US with values for the attr5, cn, and sn attributes. This example illustrates how the "m" permission can be used to limit the attributes that can be created on a new entry.

Example #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

What rights does cn=jsmith have to o=XYZ, c=US? Make(m) on attributes all attributes and Add(a) on the entry. These are sufficient permissions to create a new object, cn=New, o=XYZ, c=US with values for any desired attributes. For administrators who do not wish to limit the attributes that can be created on new entries, this example shows how a single ldapACI at the top of the domain solves the problem.
Example #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

In this example, cn=rvh,dc=att,dc=com has "rw" access to the English-
language "description" attributes of entries under dc=com,dc=demo. cn=rob,dc=sun,dc=com has "rw" rights to both English- and French-
language "description" attributes. The example demonstrates that
"Attribute Descriptions", not just "Attribute Types", can be used in the
"attr" field of an ACI. See [LangCode] for more information on the
attribute options used here.

8.4 ModDN

There are many different actions that can happen when the modDN command
are used. The following illustrates the permissions needed in order to
execute each scenario. For all examples, we will assume the role
cn=Admins is working with the following entry:

dn: cn=personA, o=Company
  cn: personA
  cn: FirstName
  sn: LastName
  objectclass: person

Example #1

Perform a modRDN only, using an existing attribute value. In this case,
we perform a modRDN and rename cn=personA, o=Company to cn=FirstName,
o=Company. The entryACI value for this entry must give the cn=Admin role
rename permission on the entry.

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

Example #2

We wanted to perform a ModRDN and add a new attribute value. In this
case we are renaming cn=personA, o=Company to cn=newFirstName,
o=Company. The entryACI value must give the cn=Admin role rename
permission on the entry and w permission on the cn attribute.

entryACI: grant:n#[entry]#authnLevel:weak:role:cn=Admin
entryACI: grant:w#cn#authnLevel:weak:role:cn=Admin
Example #3

Perform a modRDN, using an existing attribute, but delete the old RDN value. Here, we perform a modRDN and rename cn=personA, o=Company to cn=FirstName, o=Company and set the deleteOldRdn flag to true. The cn=Admin role must have permission to rename the entry, and to delete a cn attribute value (i.e. ‘o’).

entryACI: grant:n#entry#authnLevel:weak:role:cn=Admin
entryACI: grant:o#OID.cn#authnLevel:weak:role:cn=Admin

Example #4

Perform a modRDN, using an new cn attribute value (cn=newFirstName), and delete the old RDN value (cn=personA). Here, we perform a modRDN and rename cn=personA, o=Company to cn=newFirstName, o=Company and set the deleteOldRdn flag to true. The cn=Admin role must have permission to rename the entry, and to delete and write the cn attribute value.

entryACI: grant:n#entry#authnLevel:weak:role:cn=Admin
entryACI: grant:w,o#OID.cn#authnLevel:weak:role:cn=Admin

Example #5

We want to change the entry’s location in the DIT (modDN) and keep the same RDN Value. In this case we are moving cn=personA, o=Company to cn=personA, o=CompanyB. The cn=Admin role must have export permission on the original entry, and import permission on the new superior object (the new parent). The cn=personA, o=Company entryACI is:

entryACI: grant:e#entry#authnLevel:weak:role:cn=Admin

and the o=CompanyB entryACI is:

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

Example #6

We want to change the entry’s location in the DIT (modDN) and change the RDN Value to an existing attribute value. In this case we are moving cn=personA, o=Company to cn=FirstName, o=CompanyB. The cn=Admin role must have rename and export permission on the original entry, and import permission on the new superior object (the new parent). The cn=personA, o=Company entryACI is:

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

and the o=CompanyB entryACI is:

entryACI: grant:en#entry#authnLevel:weak:role:cn=Admin
Example #7

We want to change the entry’s location in the DIT (modDN) and change the RDN Value to a new attribute value. In this case we are moving cn=personA, o=Company to cn=newFirstName, o=CompanyB. The cn=Admin role must have rename and export permission on the original entry, write permission on the naming attribute (cn) and import permission on the new superior object (the new parent). The cn=personA, o=Company entryACI is:

```
entryACI: grant:en#[entry]#authnLevel:weak:role:cn=Admin
entryACI: grant:w#OID.cn#authnLevel:weak:role:cn=Admin
```

and the o=CompanyB entryACI is:

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

Example #8

We want to change the entry’s location in the DIT (modDN) and change the RDN Value to an existing attribute value, and delete the old RDN value. In this case we are moving cn=personA, o=Company to cn=FirstName, o=CompanyB and deleting the attribute value personA. The cn=Admin role must have rename and export permission on the original entry, delete (‘o’) permission on the naming attribute (cn) and import permission on the new superior object (the new parent). The cn=personA, o=Company entryACI is:

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

and the o=CompanyB entryACI is:

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

Example #9

We want to change the entry’s location in the DIT (modDN) and change the RDN Value to a new attribute value, and delete the old RDN value. In this case we are moving cn=personA, o=Company to cn=newFirstName, o=CompanyB and deleting the attribute value personA. The cn=Admin role must have rename and export permission on the original entry, write (‘w’), and delete (‘o’) permission on the naming attribute (cn) and import permission on the new superior object (the new parent). The cn=personA, o=Company entryACI is:
entryACI: grant:en#[entry]#authnLevel:weak:role:cn=Admin
entryACI: grant:wo#OID.cn#authnLevel:weak:role:cn=Admin
and the o=CompanyB entryACI is:
entryACI: grant:i#[entry]#authnLevel:weak:role:cn=Admin

8.5 Interaction Among ACI

These examples show how ACI in different parts of the tree interact. Examples with varying authnLevel are given in the next section.

The examples refer to this fragment of a directory tree:

```
dc=com
   +-----------------------+
   |                        |
   dc=tivoli                 dc=sun
   |                        |
   cn=ellen                  cn=rob
```

Example #1

If the ACI is as follows:

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

Then the effective rights of cn=rob,dc=sun,dc=com to all the attributes of object cn=ellen,dc=tivoli,dc=com are "rw". The ACI at dc=tivoli,dc=com is redundant.

Example #2

If the ACI is as follows:

```
at dc=com:  subtreeACI:grant:r#[all]# authnLevel:weak:
   authzID-dn:cn=rob,dc=sun,dc=com
at dc=tivoli,dc=com:  subtreeACI:grant:w#OID.uid#authnLevel:weak:
   authzID-dn:cn=rob,dc=sun,dc=com
```

Then cn=rob,dc=sun,dc=com has effective rights of "r" to all the attributes of object cn=ellen,dc=tivoli,dc=com, and effective rights of "rw" to the uid attribute of object cn=ellen,dc=tivoli,dc=com. Also, cn=rob,dc=sun,dc=com has effective rights of "r" to all attributes of object cn=rob,cn=sun,cn=com.
Example #3

If the ACI is as follows:

\[
\text{at dc=com: subtreeACI:grant:rw#[all]#authnLevel:weak:}
\text{authzID-dn:cn=rob,dc=sun,dc=com}
\text{at dc=tivoli,dc=com: subtreeACI:deny:w#[all]#authnLevel:weak:}
\text{authzID-dn:cn=rob,dc=sun,dc=com}
\]

Then \text{cn=rob, dc=sun, dc=com} has effective rights of "r" (but not "w") to all the attributes of object \text{cn=ellen, dc=tivoli, dc=com} and has effective rights of "rw" to all attributes of objects \text{cn=rob, dc=sun, dc=com}.

Example #4

If the ACI is as follows:

\[
\text{at dc=com: subtreeACI:grant:r#OID.uid#authnLevel:weak:}
\text{authzID-dn:cn=rob,dc=sun,dc=com}
\text{at dc=tivoli,dc=com: subtreeACI:grant:w#OID.sn#authnLevel:weak:}
\text{authzID-dn:cn=rob,dc=sun,dc=com}
\]

Then \text{cn=rob, dc=sun, dc=com} has effective rights of "r" to the uid attribute and "w" to the sn attribute of object \text{cn=ellen, dc=tivoli, dc=com}.

Example #5

If the ACI is as follows:

\[
\text{at dc=com: subtreeACI:grant:r#OID.uid#authnLevel:weak:}
\text{authzID-dn:cn=rob,dc=sun,dc=com}
\text{at cn=rob,dc=sun,dc=com: entryACI:grant:rw#[all]#authnLevel:weak:}
\text{this:}
\]

Then \text{cn=rob, dc=sun, dc=com} has effective rights of "rw" to all attributes of object \text{cn=rob, dc=sun, dc=com}.

Example #6

If the ACI is as follows:

\[
\text{at dc=com: subtreeACI:grant:rw#uid#authnLevel:weak:}
\text{authzID-dn:cn=rob,dc=sun,dc=com}
\text{at dc=tivoli,dc=com: subtreeACI:deny:w#uid#authnLevel:weak:}
\text{subtree:dc=sun,dc=com}
\]

Then \text{cn=rob, dc=sun, dc=com} has rights of "r" to the uid attribute of object \text{cn=ellen, dc=tivoli, dc=com}. While checking the "w" permission, the subtreeACI at \text{dc=tivoli, dc=com} is lower in the tree than the subtreeACI at \text{dc=com}, so it takes precedence at Step 1.
Example #7

If the ACI is as follows:

\[
\text{at } dc=com: \quad \text{subtreeACI:grant:rw#OID.uid#authnLevel:weak:} \\
\text{authzID-dn:cn=rob,dc=sun,dc=com} \\
\text{at } dc=com: \quad \text{subtreeACI:deny:w#OID.uid#authnLevel:weak:} \\
\text{subtree:dc=sun,dc=com}
\]

Then \( cn=\text{rob,dc=sun,dc=com} \) has rights of "rw" to the uid attribute of object \( cn=\text{ellen,dc=tivoli,dc=com} \). While checking the "w" permission, the two subtreeACI are at the same level in the tree (step 1) and the Subject Type \( dn-dn \) has precedence over Subject Type subtree (step 2), so the first ACI has precedence over the second.

Example #8

If the ACI is as follows:

\[
\text{at } dc=com: \quad \text{subtreeACI:grant:rw#OID.uid#authnLevel:weak:} \\
\text{subtree:dc=sun,dc=com} \\
\text{at } dc=com: \quad \text{subtreeACI:deny:w#OID.uid#authnLevel:weak:} \\
\text{subtree:dc=sun,dc=com}
\]

Then \( cn=\text{rob,dc=sun,dc=com} \) has rights of "r" to the uid attribute of object \( cn=\text{ellen,dc=tivoli,dc=com} \). While checking the "w" permission, the two subtreeACI are at the same level in the tree (step 1) and they have the same Subject Type (step 2), so the precedence of deny over grant (step 5) is the deciding factor.

Example #9

If the ACI is as follows:

\[
\text{at } dc=com: \quad \text{subtreeACI:grant:rw#OID.uid#authnLevel:weak:} \\
\text{subtree:dc=sun,dc=com} \\
\text{at } dc=com: \quad \text{subtreeACI:deny:w#[all]#authnLevel:weak:} \\
\text{subtree:dc=com}
\]

Then \( cn=\text{rob,dc=sun,dc=com} \) has rights of "rw" to the uid attribute of object \( cn=\text{ellen,dc=tivoli,dc=com} \). While checking the "w" permission, the two subtreeACI are at the same level in the tree (step 1) and they have the same Subject Type (step 2), so the precedence of a specific attribute list over "[all]" (step 4) is the deciding factor.

8.6 Use of ipAddress in ACI

Using the tree fragment from Section 8.5:
Example #1

If the ACI is as follows:

```
    at 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:
```

Then any user regardless of the strength of their authentication is

denied all permissions if they happen to be connecting from an IP

address in the 10-net. If they connect from any other address, they

have "rscp" permissions for all attributes and "btv" permissions for all

entries in the dc=com subtree.

Example #2

If the ACI is as follows:

```
    at 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
```

It will have no effect. While it might seem that it would grant total

access to any user bound from an address in 10-net, the special rules

governing ipAddress and dns as subjects make them useful only for deny

ACI, not for grants. This was done because the effects of a mistaken

grant to an IP address range or wildcared DNS name could be extremely

serious.

An (insane) administrator who really wants to grant total access to

anyone on 10-net would have to specify:

```
    at 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
```

This ACI depends on the fact that a "deny" works on the stated

authnLevel and all lower authnLevels while a "grant" works on the stated

level and all higher authnLevels.
8.7 Use of authnLevel in ACI

Using the tree fragment from Section 8.5:

Example #1

If the ACI is as follows:

```plaintext
at dc=com: subtreeACI:grant:rw#OID.sn#authnLevel:strong:  
  authzID-dn:cn=rob,dc=sun,dc=com
at dc=tivoli,dc=com: subtreeACI:grant:r#OID.sn#authnLevel:limited:  
  authzID-dn:cn=rob,dc=sun,dc=com
```

Then `cn=rob,dc=sun,dc=com` has effective rights of "rw" to the `sn` attribute of object `cn=ellen,dc=tivoli,dc=com` if the BIND for this session used strong authentication. If the BIND for this session used limited authentication, `cn=rob,dc=sun,dc=com` has effective rights of "r" to the `sn` attribute of object `cn=ellen,dc=tivoli,dc=com`. If the BIND for this session used weak or no authentication, `cn=rob,dc=sun,dc=com` has no rights to object `cn=ellen,dc=tivoli,dc=com`.

Example #2

If the ACI is as follows:

```plaintext
at dc=com: subtreeACI:grant:rw#sn#authnLevel:limited:  
  subtree:dc=sun,dc=com
at dc=tivoli,dc=com: subtreeACI:grant:c;deny:w#sn#authnLevel:strong:  
  authzID-dn:cn=rob,dc=sun,dc=com
```

Then `cn=rob,dc=sun,dc=com` has effective rights of "rc" to the `sn` attribute of object `cn=ellen,dc=tivoli,dc=com` if the BIND for this session used strong authentication. The "r" permission comes from the fact that the grant part of the first ACI applies to BINDs at or above the "limited" level. If the BIND for this session used limited authentication, `cn=rob,dc=sun,dc=com` has "r" rights to the `sn` attribute of object `cn=ellen,dc=tivoli,dc=com`. The deny part of the second ACI applies to cases where the authnLevel is less than "strong", so it overrides the grant of "w" permission in the first ACI. If the BIND for this session is at the weak or none authnLevel, the user has no permissions.

Example #3

If the ACI is as follows:

```plaintext
at dc=com: subtreeACI:grant:rs#sn#authnLevel:none:public
at dc=com: subtreeACI:grant:w#sn#authnLevel:strong:  
  subtree:cn=rob,dc=sun,dc=com
```

Then `cn=rob,dc=sun,dc=com` has effective rights of "rsw" to the `sn` attribute of object...
attribute of object cn=ellen,dc=tivoli,dc=com if the BIND for this
session used strong authentication and effective rights "rs" if the BIND
for this session used any other form of authentication. The grant in
the first ACI applies to BINDs at the "none" level and above, so it
applies to any authnLevel.

Example #4

If the ACI is as follows:

at root: subtreeACI:grant:ps#[all]#authnLevel:none:public:
    subtreeACI:grant:cr#[all]#authnLevel:weak:subtree:

Then any user including the anonymous user (no name supplied to BIND)
has "ps" permissions to any entry on the server, and any user with an ID
on the server ("subtree:" specifies any DN in the subtree under root)
who has bound using "weak" or better authentication has "pscr"
permissions.

Example #5

If the ACI is as follows:

at dc=com: subtreeACI:grant:rw#[all]#authnLevel:limited:
    cn=ellen,dc=tivoli,dc=com
at dc=tivoli,dc=com: subtreeACI:deny:w#[all]#authnLevel:strong:
    cn=rob,dc=sun,dc=com

Then if bound at the strong authnLevel, user cn=ellen,dc=tivoli,dc=com
has permissions "rw" to all the attributes of the object
cn=ellen,dc=tivoli,dc=com, and permissions "rw" to all the attributes of
the object cn=rob,dc=sun,dc=com. But if bound at the limited
authnLevel, user cn=ellen,dc=tivoli,dc=com has permissions "r" to all
the attributes of the object cn=ellen,dc=tivoli,dc=com, and permissions
"rw" to all the attributes of the object cn=rob,dc=sun,dc=com.

This is a consequence of the way that "deny" is processed with respect
to authnLevel. Since cn=rob,dc=sun,dc=com is denied "w" permission when
 authenticating at the "strong" authnLevel, ALL users are denied "w"
 permission when bound at any weaker level (see the rules for authnLevel
 in "Subjects and Associated Authentication").

9. GetEffectiveRights Control

The access control information controls provide a way to manipulate
access control information in conjunction with a LDAP operation. One
LDAP control is defined. This control allows access control information
to be retrieved. The control is:

    GetEffectiveRights - obtain the effective rights for a given
directory entry(s) for a given subject during a ldap_search operation.

The purpose of the getEffectiveRights control is to allow an administrator or application to query the server about the rights of another user of the directory. This is important as it would allow the administrator to verify the access control policy for a given subject. Or it would allow an application for example, to propose to a user the attributes which he has the right to modify or see in a given entry.

9.1 Request Control

This control may only be included in the ldap_search message as part of the controls field of the LDAPMessage, as defined in Section 4.1.12 of [LDAPv3].

The controlType is set to <OID to be assigned>. The criticality MAY be either TRUE or FALSE (where absent is also equivalent to FALSE) at the client’s option. The controlValue is an OCTET STRING, whose value is the BER encoding of a value of the following SEQUENCE:

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
}

The getEffectiveRightsRequest specifies a subject, gerSubject, about whom access control information is being requested. The control asks the server to evaluate and return the entry level rights possessed by the gerSubject for each entry that is returned in the search results, and for each returned or specifically requested attribute. The server will use the Access Decision Algorithm from section 4.3 to determine the requested effective rights; it will be seen that the parameters defining the subject in the Access Decision algorithm ((dn OPTIONAL, ipAddress, dnsName, authnLevel)) are all present in this control.
9.2 Response Control

This control is included in the ldap_search_response message as part of the controls field of the LDAPMessage, as defined in Section 4.1.12 of [LDAPv3].

The controlType is set to <OID to be assigned>. There is no need to set the criticality on the response. The controlValue is an OCTET STRING, whose value is the BER encoding of a value of the following SEQUENCE:

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

The effective rights returned are returned with each entry returned by the search result. The control response for ldap_search is:

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

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

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

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

For a given entry, the control response entryLevelRights field contains
the entry level effective rights that gerSubject has on that entry. The
attributeLevelRights field contains a list of attributes and the
effective rights that gerSubject has for each of those attributes. The
list of attributes consists of those attributes returned as part of the
search operation plus any explicitly requested attributes that were not
returned. An attribute explicitly requested in the search request might
not be returned because the attribute is not in the entry, but we may
still be interested in the effective rights on it, for example to
determine if gerSubject could add that attribute to the entry.

The control returns the permissions that gerSubject has on a given entry
and its attributes. In order to determine whether these permissions
suffice to allow gerSubject to perform a given LDAP operation on the
entry, the requestor will determine if these permissions satisfy the
required permissions for that LDAP operation, as defined in section 5.

Note that in the case where gerSubject does not have a particular
permission then this control does not allow the requestor to determine
whether that is because the permission was not granted to gerSubject or
whether it was because this permission has been explicitly denied.

The required permissions to see the effective rights of a subject on an
entry and its attributes are specified in 9.3.

If gerSubject has the same rights to a set of attributes in that entry
then the server may group the attributes together in a list.

Although this extends the search operation, there are no
incompatibilities between versions. LDAPv2 cannot send a control, hence
the above structure cannot be returned to a LDAPv2 client. A LDAPv3
client cannot send this request to a LDAPv2 server. A LDAPv3 server not
supporting this control cannot return the additional data.

9.3 Access control for the Get Effective Rights Control

In the presence of the get effective rights control, the access controls
applied to the search operation use the requestor’s authorization
identity.

For a given entry returned in the search results then in order to see
the effective rights of any subject for this entry and its attributes
the requestor requires:

1. "g" permission on the entry.

If 1. is not satisfied then the "insufficientAccess" error is returned
for the entryLevelRights of that entry and for each of the rights in the
AttributeLevelRights list.

Note A: the set of entries and attributes that are returned are those
visible to the requestor, not the gerSubject. This means that it is possible that if gerSubject could see an entry that the requestor could not then it is impossible for the requestor, using the getEffectiveRights control, to retrieve the effective rights of gerSubject on that entry. However, the idea here is that the requestor will typically be a powerful administrator whose access rights will be a superset of those of other users.

Note B: once a given subject has the "g" permission on a given entry then he has the right to see the effective rights of _any_ subject on that entry. It might be useful to have a way to restrict the set of subjects whose effective rights can be retrieved but that complicates the model in that, for the "g" permission only, we no longer have a target/subject type structure but rather a target/subject/otherSubject structure. Here, we choose the simpler model rather than this extra functionality.

9.4 Get Effective Rights example

Suppose we have a DIT with the entries and access controls as described in the following LDIF example:

```ldif
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:strong
    group: 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
```

Stokes, et al Expires 29 December 2001 [Page 54]
The access control policy in this DIT policy may be described as follows:

1. anonymous users have full read, search, and compare rights to the whole DIT, except for the important attributes userPassword, subtreeACI, entryACI, and salary.

2. Any user bound with a limited authentication level can modify any attributes in his own entry, except subtreeACI, entryACI and salary.

3. Any user can read all attributes in his own entry.
4. Any user bound with a limited authentication level can retrieve the effective rights on his own entry (including userPassword, salary, entryACI and subtreeACI).

5. users, bound with a strong authentication level, in the cn=adminGroup,ou=Groups,o=sun.com group have full rights in the whole DIT.

Then here are some examples of requests to get the effective rights and the responses:

Example 1. Suppose we issue a search authenticated to level strong as "cn=admin,o=sun.com" (who is in the group cn=adminGroup,ou=Groups,o=sun.com), with base "o=sun.com", search filter "objectclass=*", requested attributes "* entryACI", with the getEffectiveRights control subject set to "cn=Joe Sales,ou=Sales,o=sun.com" and the MachineSubject specifying the ipAddress and dnsName of the client machine and the authnLevel set to limited.

Then the search result and effective rights we see are:

```
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

---
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

9.5 Client-Server Interaction

The GetEffectiveRightsRequest control requests the rights that are in effect for requested directory entry/attribute based on the specified subject. The server that consumes the search operation looks up the rights for the returned directory information based on the subject and returns that rights information as defined above.
There are six possible scenarios that may occur as a result of the GetEffectiveRights control being included on the search request:

1. If the server does not support this control and the client specified TRUE for the control’s criticality field, then the server MUST return unavailableCriticalExtension as a return code in the searchResponse message and not send back any other results. This behavior is specified in section 4.1.12 of [LDAPv3].

2. If the server does not support this control and the client specified FALSE for the control’s criticality field, then the server MUST ignore the control and process the request as if it were not present. This behavior is specified in section 4.1.12 of [LDAPv3].

3. If the server supports this control but for some reason the server cannot process it and the client specified TRUE for the control’s criticality field, then the server SHOULD do the following: return unavailableCriticalExtension as a return code in the searchResult message.

4. If the server supports this control but for some reason the server cannot process it and the client specified FALSE for the control’s criticality field, then the server should process as 'no rights returned' and include the result Unavailable in the GetEffectiveRightsResponse control in the searchResult message.

5. If the server supports this control and can return the rights information, then it should include the GetEffectiveRightsResponse control in the searchResult message with a result of success.

6. If the search request failed for any other reason, then the server SHOULD omit the GetEffectiveRightsResponse control from the searchResult message.

The client application is assured that the correct rights are returned for scope of the search operation if and only if the GetEffectiveRightsResponse control returns the rights. If the server omits the GetEffectiveRightsResponse control from the searchResult message, the client SHOULD assume that the control was ignored by the server.

The GetEffectiveRightsResponse control, if included by the server in the searchResponse message, should have the GetEffectiveRightsResult set to either success if the rights are returned or set to the appropriate error code as to why the rights could not be returned.

The server may not be able to return a right because it may not exist in that directory object’s attribute; in this case, the rights request is ignored with success.
10. Security Considerations

This document proposes protocol elements for transmission of security policy information. Security considerations are discussed throughout this model. Because subject security attribute information is used to evaluate decision requests, it is security-sensitive information and must be protected against unauthorized modification whenever it is stored or transmitted.

Interaction of access control with other directory functions (other than the ones defined in this document) are not defined in this document, but instead in the documents where those directory functions are defined. For example, the directory replication documents should address the interaction of access control with the replication function.

In general, IP addresses and DNS names should not be used as identities (subjects) since they can be easily spoofed. However, some widely-deployed implementations have long supported and continue to support IP addresses and DNS names in ACI to enforce access controls based on topology. Thus IP address and DNS name are retained in the access control model though their use should be discouraged in new deployments.

It is good security practice to set defaults to the most secure settings. This is done to ensure that accidentally omitting a security field does not compromise security. Following this practice in the case of authnLevel would result in a default of "strong". This would have meant that ACI with omitted authnLevel in directories where "strong" authentication is not available (the great majority of environments at this time) would deny all access, causing confusion among users and administrators. To avoid this problem, authnLevel is required on all ACI. This has the useful side-effect of forcing administrators to think about the strength of their authentication system when setting up ACI.

All of the advantages of authnLevel-based access control may be lost if system administrators do a poor job of associating actual authentication mechanisms with the authnLevels in the model. Administrators should use external guidelines (for an example, see [AUTHMAP]) if they are not completely familiar with the relative strengths of the authentication mechanisms available in their environment. In addition, administrators in replicated environments should make sure that the authnLevel/authentication mechanism mappings at all replicating sites are consistent.

11. References


ACKNOWLEDGEMENT

This is to acknowledge the numerous companies and individuals who have contributed their valuable help and insights to the development of this
specification.

AUTHOR(S) ADDRESS

Ellen Stokes                       Bob Blakley
Tivoli Systems                     Tivoli Systems
9442 Capital of Texas Hwy N        9442 Capital of Texas Hwy N
Austin, TX 78759                   Austin, TX 78759
USA                                USA
mail-to: estokes@tivoli.com        mail-to: blakley@tivoli.com
phone: +1 512 436 9098             phone: +1 512 436 1564
fax:   +1 512 436 1190             fax:   +1 512 436 1199

Debbie Rinkevich                   Robert Byrne
Momenta                            Sun Microsystems
---                                14 Chemin du Vieux Chene
Austin, TX                         Meylan ZIRST 38240
USA                                France
mail-to: drinkevich@momenta.com    mail-to: robert.byrne@.sun.com
phone: +1 512 732 0060  ext 125    phone: +33 (0)4 76 188 205

Rick Huber
AT&T Laboratories
200 Laurel Avenue South
Middletown, NJ  07748
USA
mail-to: rvh@att.com
phone: +1 732 420 2632
fax:   +1 732 368 1690