OpenDS

Access Control Requirements

Version 0.1
December 2006

1. Contents

2. Overview

The access control subsystem is one of the most critical components of a directory server. It helps ensure that sensitive information is available only to those who need it, and that only the appropriate users may perform restricted operations. Historically, directory servers are responsible for protecting information and users that exist entirely within the local server, but those requirements are changing and access control models must change along with them. This document discusses the requirements that the OpenDS access control model must meet, as well as additional features that should be considered in its design.

Note that this document attempts to stay entirely within the realm of a requirements specification. It does not intend to suggest any particular implementation. The OpenDS access control model should meet the requirements laid forth in this specification, but the underlying design and the interfaces used to expose it are outside the scope of this document.

2.1. Copyright and trademark information

2.2. Feedback

Please direct any comments or suggestions about this document to:
dev@opends.dev.java.net

2.3. Acknowledgements

The general format of this document was based on the documentation template used by OpenOffice.org.

2.4. Modifications and Updates

Date	Description of Change
November, 2006	Initial draft

3. Access Control Feature Summary and Prioritization

This document contains a large number of requirements and suggested functionality for the OpenDS access control implementation. Due to the size of the document and the way that some of the requirements may be embedded in prose text, it is possible that some of them may be overlooked. To help with this issue, this section provides a concise summary of all of the access control features referenced throughout this document. For the casual reader that wants to understand the set of access control features that we intend to provide, it may be sufficient to only read this section.

This section also attempts to assign priorities to the access control features. This makes it possible to understand their relative importance, as well as the level of assurance that they may be implemented in OpenDS. Note, however, that this document may be in flux and the set of features and their associated priorities may change over time. The priorities that will be used include:

- Must Have (MH) -- These features are those which are absolutely required for the access control implementation. Many of them are required to satisfy requirements for backward compatibility, migration from the Sun Java System Directory Server, or standards compliance. Others are features that we simply feel are necessary to include in the implementation to provide the best level of protection for the directory data. Any commercial release of a Sun product based on OpenDS must include these features.

- Highly Desirable (HD) -- These features are those which are very important to include in OpenDS, but are not so important as to be considered absolute requirements. It is likely that they will be included in OpenDS, but if necessary they may be allowed to slip such that they may not be included in the first commercial release of a Sun product based on OpenDS.

- Nice to Have (NTH) -- These features are those which may provide some level of benefit but are not critical to the OpenDS implementation. They may be included in the product if time permits, but priority must be given to the "must have" and "highly desirable" features.

- Not Planned (NP) -- These features are those which may be mentioned in this document but we do not currently intend to implement in OpenDS for some reason.

The remainder of this section will describe the features in the order in which they are listed in the rest of the document along with their priorities. Links will be provided to the section describing them in more complete detail for additional information.

- [Section 4.1] (MH) -- OpenDS must ensure that all access control rules are expressed using the correct syntax and are free of ambiguity. It must identify an parse all access control rules for correctness at startup. It must reject any attempt to add or modify an access control rule in a manner that the resulting rule cannot be parsed. It must reject any attempt to delete an access control rule on which other rules are dependent.

- [Section 4.1] (MH) -- OpenDS must provide the ability to start in a mode in which the server will only allow clients to connect over the loopback interface and will only allow operations by root users. It must be possible for an administrator to place the server in this mode at any time, and the server should place itself in this mode if it detects any inconsistencies at startup.

- [Section 4.1] (HD) -- When processing an LDIF import, OpenDS should attempt to include access control rules that it may not be completely possible to parse. The server must detect these invalid rules at startup and enter a maintenance mode so that the administrator may manually correct them.

- [Section 4.2] (MH) -- OpenDS must provide a migration path for access control rules in deployments based on the Sun Java System Directory Server 5.2 and 6.0. This may be implemented through direct support for the current access control syntax or some kind of translation mechanism.

- [Section 4.2] (NTH) -- OpenDS may provide a migration path for access control rules in LDAP directory deployments other than those based on the Sun Java System Directory Server. It may also be useful to consider migrations from other types of products like X.500 directories, relational databases, or RADIIUS/DIAMETER servers.

- [Section 4.2] (NP) -- We do not intend to implement the access control syntax described in the Access Control Model for LDAPv3 Internet Draft. We also do not intend to provide complete support for all elements listed in RFC 2820.

- [Section 4.3] (HD) -- OpenDS should provide a mechanism that will allow for it to detect and apply changes to the access control configuration of an instance of the Sun Java System Directory Server during a migration period when both OpenDS and the Sun Java System Directory Server may coexist.

- [Section 4.4] (MH) -- The OpenDS access control implementation must provide the highest possible level of performance and scalability while still ensuring completely correct behavior.

- [Section 4.5] (MH) -- The OpenDS access control implementation must provide a mechanism for restricting access to data contained in remote data sources, particularly proxied, distributed, and virtual backends.

- [Section 4.6] (MH) -- The OpenDS access control implementation must provide a mechanism for storing access control rules separately from the data to which they apply.

Note that it may still be permissible for access control rules to be embedded in the server data, but it must not be required (especially for data contained in remote data sources, like proxied, distributed, and virtual backends).

- [Section 4.7] (MH) -- It must be possible to scope access control rules so that they apply to a single entry, a particular subset of entries, or all entries in the server. There should be a number of mechanisms for identifying subsets of entries.

- [Section 4.7] (NTH) -- OpenDS may provide support for LDAP subtrees as defined in RFC 3672 to identify entries that fall in the scope of an access control rule.

- [Section 4.8] (HD) -- OpenDS should provide mechanisms for delegating portions of access control management. This should include organizational administrators defining a set of rules to apply to their organization, managers with some level of control over what may be visible to their subordinates, and individuals with some level of control over what may be visible to others.

- [Section 4.9] (MH) -- OpenDS must provide a mechanism for synchronizing access control rules across multiple instances.

- [Section 4.9] (HD) -- OpenDS should provide a mechanism for selectively synchronizing access control rules across multiple instances. It should be possible to have rules that are not synchronized, and others which may be synchronized with only a specified subset of the servers in the environment.

- [Section 4.10] (MH) -- If it is possible that multiple access control rules could conflict with each other, then OpenDS must provide a clear and predictable algorithm for defining the precedence in which the rules will be evaluated.

- [Section 4.11] (MH) -- Changes to the access control configuration must take effect immediately, for both existing and future client connections.

- [Section 4.12] (MH) -- OpenDS must provide a mechanism for protecting access to virtual attributes.

- [Section 4.12] (HD) -- OpenDS should provide a mechanism that allows the values of virtual attributes to be included when making access control decisions.

- [Section 4.13] (HD) -- OpenDS should provide a mechanism for extending the functionality of the access control subsystem, and ideally to provide a public API that exposes this extensibility.

- [Section 4.14] (NTH) -- OpenDS may provide an alternate representation for access control rules which may be easier to interpret by external clients, and potentially could be used for clients to define new access control rules in the server.

- [Section 4.15] (MH) -- The OpenDS access control subsystem must remain free of external dependencies in order to ensure the greatest degree of control over our implementation and our ability to address and correct problems and meet future requirements.

- [Section 4.16] (NP) -- We do not intend to implement the access control subsystem in a manner which will make it suitable for use as a general-purpose policy engine.

- [Section 4.17] (MH) -- It must be possible to disable the OpenDS access control subsystem (eliminating any associated performance penalty) if the functionality that it provides is not required.

- [Section 4.18] (MH) -- The OpenDS access control subsystem must use an open development model and must provide a mechanism for users and other interested parties to provide feedback and potentially participate in its implementation.

- [Section 5.1] (HD) -- OpenDS should provide a syntax that is easier to use and understand, as well as easier for the server to parse, than that used by the Sun Java System Directory Server.

- [Section 5.2] (NTH) -- OpenDS may provide a mechanism for defining named sets of rights that may be applied as a single entity rather than explicitly naming all associated rights (e.g., the "all" set in the Sun Java System Directory Server).

- [Section 5.3] (MH) -- The OpenDS access control mechanism must not automatically include operational attributes when indicating which attributes should be included in the scope of a target rule. It must be possible to explicitly target operational attributes if necessary.

- [Section 5.3] (HD) -- OpenDS should provide a mechanism for specifying target attributes in access control rules based on the objectclass that includes them, similar to the mechanism described in RFC 4529.

- [Section 5.4] (HD) -- OpenDS should attempt to avoid the ambiguity associated with the use of wildcards in DNs in the Sun Java System Directory Server.

- [Section 5.5] (HD) -- OpenDS should provide an enhanced proxied authorization control which may provide a means of including information not currently available in the proxied authorization control defined in RFC 4370.

- [Section 6.1] (MH) -- OpenDS must provide a number of mechanisms for identifying the set of users that should be allowed (or not allowed) to perform a given operation, including specific user DNs, group DNs, dynamic selection criteria, the user whose entry is being targeted, only anonymous users, any authenticated users, etc.

- [Section 6.2] (MH) -- OpenDS must provide a number of mechanisms for identifying the set of operations that clients will be allowed (or not allowed) to perform, including connect to the server, requesting standard operations over protocol, including controls in the request, retrieving specific content from entries, and perform administrative tasks.

- [Section 6.3] (MH) -- OpenDS must provide the ability to define additional criteria to use in the access control evaluation process not directly related to the user identity, including the time of the request, the address of the client, the protocol used to communicate with the server, the presence of a secure communication channel, etc.

- [Section 7] (MH) -- OpenDS must provide the ability to have multiple root users. It must be possible for root users to take advantage of strong authentication mechanisms, to be subject to password policy enforcement, and to be subject to bind-based access controls.

- [Section 7.1] (MH) -- OpenDS must provide a mechanism for assigning administrative privileges to users. It must be possible to define privilege sets which may be assigned to users to more easily indicate which privileges they should have.

- [Section 7.2] (HD) -- OpenDS should provide a mechanism for scoping privileges so that it is possible to grant them for use only in a particular subtree or in other limited ways.

- [Section 7.4] (HD) -- OpenDS should provide a mechanism for creating accounts that cannot directly authenticate to the server, but that may only be assumed by authorized users.

- [Section 8.1] (MH) -- OpenDS must provide support for the use of legacy static groups (based on `groupOfNames` and `groupOfUniqueNames`) and dynamic groups (based on `groupOfURLs`) for use in access control evaluation. It must be possible to support limited nesting when using static groups.

- [Section 8.2] (MH) -- OpenDS must provide support for an enhanced grouping mechanism which makes it possible to combine dynamic membership criteria, explicit include and exclude member lists, and to nest other static, dynamic, and enhanced groups.

- [Section 8.2] (NP) -- We do not intend to implement support for the LDAP: Dynamic Groups for LDAPv3 Internet Draft that describes Novell's implementation for dynamic groups.

- [Section 8.3] (MH) -- OpenDS must provide support for a virtual attribute that appears in user entries and indicates the set of groups in which that user is a member.

- [Section 8.4] (HD) -- OpenDS should provide a mechanism that makes it possible to create a virtual static group based on a dynamic or enhanced group definition.

- [Section 8.5] (HD) -- OpenDS should provide controls which may be used to determine whether a given user is a member of a specified group, the set of groups in which a given user is a member, and the set of members for a specified group. It should also provide extended operations that may be used to add members to or remove members from static and enhanced groups.

- [Section 8.6] (MH) -- The OpenDS access control subsystem must provide the ability to react immediately to changes involving membership in groups referenced by one or more access control rules.

- [Section 8.7] (MH) -- OpenDS must provide a mechanism for maintaining referential integrity in group membership, access control rules, and other areas.

- [Section 8.8] (MH) -- OpenDS must provide the ability to have legacy static groups with no members.

- [Section 8.9] (NP) -- We do not intend to provide support for roles as implemented in the Sun Java System Directory Server. We will provide support for all functionality that roles provided through other grouping mechanisms, and there will be a migration path away from roles.

- [Section 9] (MH) -- The OpenDS access control mechanism must provide at least minimal capability for rejecting client connections. It must be possible to place the server in a mode in which it will only accept client connections on the loopback interface, and will only allow operations from root users.

- [Section 9] (HD) -- OpenDS should migrate support for rejecting client connections from the connection handlers into the access control subsystem.

- [Section 10.1] (HD) -- OpenDS should provide a mechanism to restrict the use of abandon operations in the case that the authorization DN of the abandon request does not match the authorization DN for the target operation.

- [Section 10.2] (MH) -- OpenDS must provide a mechanism for restricting access for add operations based on the contents of the entry to add.

- [Section 10.3] (MH) -- OpenDS must provide a mechanism for restricting access for bind operations based on the protocol version, the authentication DN, the authorization DN, the authentication type, the SASL mechanism, the authentication and/or authorization DN for the client before beginning the modify operation, and the contents of the target user entry.

- [Section 10.4] (MH) -- OpenDS must provide a mechanism for restricting access for compare operations based on the contents of the target entry, and/or the attribute type and assertion value from the compare request.

- [Section 10.5] (MH) -- OpenDS must provide a mechanism for restricting access for delete operations based on the contents of the target entry.

- [Section 10.6] (MH) -- OpenDS must provide a mechanism for restricting access for extended operations based on the request OID. Any internal operations performed in the course of processing the extended request must also be protected.

- [Section 10.6] (NTH) -- OpenDS may provide enhanced access control support for extended operations beyond restrictions based on the request OID.

- [Section 10.7] (MH) -- OpenDS must provide a mechanism for restricting access for modify operations based on the contents of the target entry (before or after the changes) and the types of changes to apply.

- [Section 10.8] (MH) -- OpenDS must provide a mechanism for restricting access for modify DN operations based on the contents of the modify DN request and/or the target entry, as well as whether the target entry has any subordinates.

- [Section 10.9] (MH) -- OpenDS must provide a mechanism for restricting access for search operations based on the contents of the search request and the base entry. It must provide a mechanism for preventing search result entries and/or references to be returned, and for altering the contents of search result entries and/or references before they are returned.

- [Section 10.10] (HD) -- OpenDS should provide a mechanism to restrict the use of unauthorized controls in an unbind request.

- [Section 10.11] (MH) -- OpenDS must provide the ability to enforce access controls for internal operations, but it must also be possible to perform internal operations for which no access control evaluation is necessary.

- [Section 11.1] (MH) -- OpenDS must provide the ability to enforce restrictions on the contents of the filter used as part of the LDAP assertion control.

- [Section 11.2] (MH) -- OpenDS must provide the ability to enforce restrictions on the entry returned as part of the LDAP pre-read and post-read controls.

- [Section 11.3] (HD) -- OpenDS should provide a mechanism to restrict the set of sort attributes that may be used with the server-side sort control.

- [Section 11.4] (MH) -- OpenDS must provide restrictions on the authorization IDs that may be used in conjunction with the proxied authorization control.

- [Section 12] (HD) -- OpenDS should provide a mechanism to restrict the set of backends that may be targeted for an LDIF import operation, as well as whether to append or replace the contents of the database and whether to allow bypassing schema checking.

- [Section 12] (HD) -- OpenDS should provide a mechanism to restrict the set of backends that may be targeted for an LDIF export operation.

- [Section 12] (HD) -- OpenDS should provide a mechanism to restrict the set of backends that may be targeted for backend backup and restore operations.

- [Section 12] (HD) -- OpenDS should provide a mechanism to restrict the set of backends, attribute types, and index types that may be targeted for index creation, regeneration, and verification operations.

- [Section 12] (HD) -- OpenDS should provide a mechanism to restrict the use of the shutdown and restart tasks based on the presence of a shutdown reason and whether it is a shutdown or restart operation.

- [Section 12.1] (MH) -- OpenDS must provide a mechanism for restricting the ability to invoke external tasks with the server offline through remote administrative interfaces.

- [Section 12.1] (NTH) -- OpenDS may provide a mechanism to require authentication for offline administrative operations.

- [Section 13.1] (MH) -- OpenDS must provide a mechanism for debugging the operation of the access control subsystem.

- [Section 13.2] (MH) -- OpenDS must provide support for the GetEffectiveRights control in the same manner in which it is implemented in the Sun Java System Directory Server.

- [Section 13.3] (HD) -- OpenDS should provide a mechanism for determining the set of access control rules that are evaluated for any given operation.

- [Section 13.4] (MH) -- OpenDS must provide auditing capabilities for operations performed in the server.

- [Section 13.5] (HD) -- OpenDS should provide a mechanism for performing additional processing whenever certain access control rules are processed.

- [Section 13.6] (MH) -- OpenDS must not expose sensitive information through its auditing and debugging mechanisms, and it must ensure that these features are only allowed for authorized users.

- [Section 14] (MH) -- OpenDS must provide GUI and CLI mechanisms for administering access control rules in a secure manner.

- [Section 14] (MH) -- OpenDS must provide the ability to read and write access control rules using generic LDAP clients using a syntax that can be easily represented in LDIF.

- [Section 14] (MH) -- OpenDS must provide administrative interfaces for managing root user accounts.

- [Section 14] (MH) -- OpenDS must provide administrative interfaces for enabling and disabling access control debugging capabilities.

- [Section 14] (MH) -- OpenDS must provide administrative interfaces for viewing and analyzing access control debug and audit data.

- [Section 14] (HD) -- OpenDS should provide a mechanism that attempts to detect any potential problems or inconsistencies in the access control policy.

- [Section 14] (HD) -- OpenDS should provide administrative interfaces for managing groups, as well as providing an interface to use the group management controls.

- [Section 14] (HD) -- OpenDS should provide administrative interfaces for displaying the effective rights that a user has when interacting with a specified entry.

- [Section 14] (HD) -- OpenDS should provide administrative interfaces for displaying the set of access control rules that are evaluated when processing a particular operation in the server.

- [Section 15] (HD) -- OpenDS should consider proposing features for standardization, like enhanced groups, the group management controls and extended operations, the enhanced proxied authorization control, and the get applicable ACIs control.

4. General Access Control Requirements

This section describes the overall requirements that must be considered when designing the OpenDS access control subsystem. Any implementation that is to be considered for use in OpenDS must address the elements listed in this section.

4.1. Correct Enforcement and Fail-Safe Degradation

Above all else, the access control subsystem must function properly. It must allow users to perform operations that are granted to them, and must reject operations that the users are not allowed to request. If the access control subsystem decides that a requested operation should not be allowed, then there must not be any change to any data in the server, whether in memory or persisted to long-term storage, nor may there be any change in state to a client session (other than the possibility of terminating the client session if certain types of disallowed operations are requested).

The access control subsystem must also ensure that all access control rules must provide correct syntax and be free of ambiguity. This should be enforced in a number of ways:

- When the Directory Server is started, it must identify and parse all access control rules defined in the server. If any problems are detected, an administrative alert must be generated and the server must be placed in an administrative mode (based on a fallback access control policy) in which only root users will be granted access over the loopback interface so that the problem may be corrected. The server must be able to validate all access control rules before it may be made available for general use.

- Any attempt to add a new access control rule that cannot be completely parsed must be rejected.

- Any attempt to modify an existing access control rule in a way such that the resulting rule could not be completely parsed must be rejected.

- Any attempt to delete an access control rule on which one or more other rules are dependent must be rejected.

- When altering the set of defined access control rules through an LDIF import, if an access control rule included in the import cannot be parsed, then the server must generate an administrative alert about the problem. If the server is offline, then it should attempt to add the invalid access control rule anyway (provided that it is possible to do so) so that it may be detected on startup to allow an administrator to correct the problem. If the server is online, or if the affected access control rule cannot be imported, then the import must be aborted and the invalid access control rule must be corrected in the LDIF data and the import restarted.

4.2. Migration from Existing Directories

In many cases, OpenDS deployments will be the result of migrations from other directories. Because the access control policy is a critical part of the directory, it is highly desirable to provide a migration path so that it is not necessary for administrators to recreate their current policy. Note, however, that in many cases it may be beneficial for administrators to recreate their current policy based on OpenDS capabilities in order to take advantage of new capabilities or consolidate multiple rules to create a smaller and more efficient policy.

The primary points of focus for migration should be the Sun Java System Directory Server versions 5.2 and 6.0. We believe that a large percentage of deployments currently running these versions will investigate migrating to OpenDS, and it is important to ensure that this migration process goes smoothly.

At a minimum, the OpenDS access control implementation must provide support for all access control features of the Sun Java System Directory Server. This does not necessarily require the same underlying implementation, nor the same syntax to express those rules. However, if it is possible to devise an access control policy in the Sun Java System Directory Server, it must also be possible to define a policy in OpenDS that is equivalent to or a superset of that policy. This may be implemented in any of several ways, including:

- Allowing access control rules to be defined using the `aci` attribute using the same syntax as the Sun Java System Directory Server. This implementation would provide the greatest degree of flexibility and the easiest migration path, although if this is the only syntax that is supported then it could adversely impact the long-term manageability of the server.

- An online translation mechanism could be used to automatically detect the addition of new access control rules (either via LDAP add operations or in LDIF imports) and dynamically translate those rules to an equivalent rule using the OpenDS native implementation. This would allow for a flexible migration path and compatibility with applications that attempt to dynamically update the access control rules in the server, but ensuring that the translation always results in an exactly equivalent policy may be complicated to ensure. If this mechanism is chosen, then it may be useful to provide a way to "quarantine" new access control rules created in this way until they have been approved by an administrator.

- An offline translation mechanism could be used to examine the access control rules in a running Sun Java System Directory Server and/or in an offline LDIF export to produce an access control policy in the native OpenDS syntax that may be added or imported into OpenDS. This model would likely be the easiest to implement and does have the advantage of allowing administrators to review the migrated policy before it is applied to the server, but it also introduces the possibility of incompatibility with applications that expect to be able to dynamically update the server's access control policy using the syntax supported by the Sun Java System Directory Server.

It may also be desirable to provide a migration path from other directory implementations, although the effort required to implement such a migration path should be given a lower priority. In many cases, a document describing the migration process may be sufficient. If any effort is to be expended in this area, then the top priority should be to ensure that the OpenDS access control model supports all features offered by such implementations in order to allow for the possibility of a manual migration. Directory server implementations to be investigated may include:

- Apache Directory Server
- CA eTrust Directory
- IBM Tivoli Directory Server
- Microsoft Active Directory
- Novell eDirectory
- Oracle Internet Directory
- OpenLDAP
- Red Hat Fedora Directory Server

Note that there are some public specifications for access control in directory servers. In particular, this includes RFC 2820 and the Access Control Model for LDAPv3 Internet Draft. In general, these specifications should not be given significant consideration in the OpenDS implementation. RFC 2820 is an informational specification that seems poorly designed and contains a number of elements that are irrelevant to (and in some cases potentially undesirable for) the OpenDS, and the Access Control Model for LDAPv3 draft never received any significant traction and is in many ways based on a "least common denominator" design. In both cases, the important elements of these specifications have been taken into consideration in writing this OpenDS Access Control Requirements specification.

Community involvement would certainly be invaluable in investigating the access control migration path from these and other implementations. It is very likely the case that directory administrators that ma be part of the OpenDS community have significantly more experience with these directory products than the core OpenDS developers, and any assistance or direction that they could provide should be welcomed.

It may also be the case that we need to also provide some consideration for migration from data stores other than LDAP directories. In particular, it may be worthwhile to investigate access control capabilities offered by X.500 directories, relational databases, and RADIUS and/or DIAMETER deployments. In these instances, the data migration path will generally be more complex, and it is therefore reasonable to assume that a manual migration of the access control policy should be sufficient.

4.3. Coexistence with Existing Directories

The process of migrating an access control policy from an existing directory deployment to OpenDS may generally be considered a one-time event. A snapshot of that policy is taken from the source directory and is applied to OpenDS. While there may be significant complexity in its implementation, it is generally possible for administrators to perform this migration in a test environment and/or at a time in which access to the production environment is limited. However, it practice it is unlikely to assume that migration of the directory deployment can be an atomic event. It will almost certainly be the case that there will be a need for both deployments to be active for a period of time, and in that case consideration should be given to changes that may be made to the access control policy in either directory deployment to ensure that it is applied equally in the other.

In most directory deployments, changes to the access control policy are rare and therefore this may not be a significant concern. It may be entirely reasonable, especially in an initial implementation, to require that administrators manually synchronize access control policy changes in both environments. Any attempt to automate the process of detecting and migrating access control changes during this coexistence phase should give priority to the Sun Java System Directory Server.

4.4. Performance and Scalability

Because access control must be enforced for virtually every operation processed within the server, it is very important to ensure that access control evaluation has as little impact on performance as possible. It is also very important to ensure that access control evaluation have little or no impact on the overall scalability of the server. Locking must be kept to an absolute minimum, and should be completely eliminated if possible. Because changes to the access control policy should be rare, it is acceptable to make the process of updating access control rules relatively expensive in order to ensure that their evaluation is as fast as possible.

The performance of the access control implementation may be addressed in a number of ways:

- It is very that the fewer access control rules that are defined in the server, the faster the implementation will be, so it will likely be worthwhile to expend effort investigating ways that the desired access control policy can be implemented using as few rules as possible. This will also provide the added advantage of ensuring that it will be easier to keep such rules in memory so that no database access or rule parsing will be required in the course of access control evaluation.

- Although reducing the total number of access control rules in the server can be beneficial, it is even more important to minimize the number of rules that must be evaluated for any particular operation. If it is possible to optimize the internal representation of such rules so that only the minimal set of rules that apply to a given operation can be identified, then the evaluation process can be made more efficient.

- If possible, the access control implementation should be designed so that it is possible to short-circuit the evaluation process once it is known with certainty whether the target operation should be allowed or denied. If this is possible, then it may also be possible to internally reorder the evaluation so that rules which are more expensive to process are evaluated last so that they can be avoided during cases in which the determination can be made without them.

- The access control implementation will likely receive a significant benefit from caching so that the need to retrieve rules from stable storage can be minimized, as well as the need to repeatedly parse such rules at the time they are needed to evaluate whether a given operation should be allowed. This caching may also apply to areas that are only marginally affiliated with access control parsing (e.g., information about group membership). Note that the entry caching capabilities of the server do make it possible to attach objects to a cached entry, and it may be worthwhile in some cases to consider making use of that facility, although the access control implementation must not introduce any dependencies on the entry caching subsystem and it must be possible for the access control implementation to operate in cases in which there is no entry cache defined in the server.

- Some aspects of the access control implementation offered by the Sun Java System Directory Server are known to be expensive to evaluate. This is particularly true for features like macro ACIs and the use of wildcards in DNs. In order to ensure ease of migration, we will need to support equivalent functionality in OpenDS, and it may also be the case that the process of implementing other types of functionality may require expensive evaluation. In these cases, it should be possible to isolate that expensive logic so that the performance penalty is minimized for access control policies that do not make use of such features.

4.5. Access Control for Remote Data

When it was originally designed, the access control model used by the Netscape Directory Server (which ultimately became the basis for the Sun Java System Directory Server) was based on the concept that the Directory Server always had the entire set of knowledge required to make an access control decision. All of the access control rules were defined in the server, and all the users, groups, and other entries involved in access control evaluation were local. However, this is no longer a safe assumption in OpenDS. There are a number of instances in which the directory data may not be local, including chaining, proxied operations, data distribution, and virtualization. These possibilities create a number of challenges that need to be addressed in the access control implementation:

- Backends that perform simple chaining or proxy operations pass requests for parts of the directory content via LDAP to another directory server.

- Data distribution introduces an interesting problem in which no directory instance contains the entire set of data, and it is possible for a user whose entry exists in one data set to request an operation on an entry that exists in another data set. It is also possible in these cases that additional information needed by the access control evaluation process (e.g., a group definition) to exist in a third data set.

- Data virtualization introduces another interesting problem in that any given entry may be comprised of data pulled from multiple data sources, so that no one backend repository may necessarily have the entire contents for an entry.

- Data virtualization also includes a constraint that not all of the backend directory instances may be LDAP directory servers. Even for cases in which a backend server is an LDAP directory, that server may not support all of the same kinds of controls and other capabilities offered by OpenDS.

- Remote repositories accessed by OpenDS may have their own access control implementations that need to be considered. While the simplest approach may be to allow OpenDS to perform all access control evaluation and to ensure that it authenticates to all remote repositories with an account that has sufficient permissions to do anything that might legitimately be requested through the server, we at least need to account for the potential impact on auditing capabilities within that remote repository.

The OpenDS access control model must ultimately be able to address all of these concerns. If it is possible to express information about the true identity of the requester to the backend repository while still allowing the correct access control enforcement, then an attempt should be made to do so. For those backend repositories that are OpenDS server instances, we may need to consider the development of custom controls, extended operations, and/or chaining protocols so that we can ensure that the server has all necessary information to make the appropriate decisions.

4.6. Access Control Rule Placement

In the Sun Java System Directory Server, access control rules have traditionally been placed with the data to which they apply. An access control rule placed in an entry may apply to that entry and any of its subordinates. The OpenDS access control implementation may choose to allow access controls to be placed in the data (and this will be required if the implementation attempts to mirror that of the Sun Java System Directory Server for the purposes of migration), it is no longer sufficient. In OpenDS, it must be possible to define access control rules in a location outside of the data that they protect. This is necessary to provide access control for remote data in distributed and/or virtual environments. In distributed environments, any given backend server may not have all of the information necessary to make access control decisions. This is also true for virtual environments, but there is the added constraint that it may not be possible to define access control rules in the underlying data stores.

It will likely be the case that OpenDS will require a custom backend designed to store access control rules. This will be necessary at a minimum for data contained in distributed and/or virtual backends, but it may also be beneficial for storing access control rules for local data. This may have a number of benefits, including:

- It will be possible to optimize the data store for holding access control rules. It will not necessarily be the case that the access control rules be stored as entries, which means that they can be optimized for efficient loading and parsing. Appropriate indexes can be defined for quick access to the content in a number of ways.

- It will be easy to find all of the access control rules when the server is started. On startup, the server must locate and validate all access control rules, and this is significantly easier if they are all stored in one place than if they are scattered throughout all of the server backends.

- It will be easier to handle significant administrative changes once the server is online. If, for example, a backend is reinitialized with new content after the server has started, then it may contain changes to access control rules that require the access control subsystem to detect and apply. If all of the access control rules are contained in a single location dedicated to the purpose of storing such rules, then there is no such problem unless the access control data store is reinitialized (which will likely not be done with the server online for security reasons).

- It is possible to store additional information with the access control rules, including metadata that may be used for delegation of access control management and to control if and how access control rules should be synchronized between instances. These topics will be addressed in more detail later in this section.

Using a custom data store for access control rules should not in any way interfere with the ability to define the portions of the directory data to which they apply.

4.7. Access Control Scoping

In the Sun Java System Directory Server, access controls have historically been placed in an entry and applied to that entry and all entries below it. This scope may be altered somewhat by defining a target and/or target filter to provide additional criteria for identifying which entries in particular fall under the jurisdiction of the rule, but there is very little ability to restrict the scope of an access control rule. The OpenDS access control implementation must allow a great deal of flexibility in indicating exactly which entries are within the scope of a given access control rule. The access control implementation must allow for all of the following:

- It must be possible to define an access control rule that applies to all entries in the server, regardless of their position in the hierarchy.
- It must be possible to define an access control rule that applies to all entries at or below a given location in the hierarchy.
- It must be possible to define an access control rule that applies only to a single entry.
- It must be possible to define search filters that may be used to identify whether a given entry should be included in or excluded from the scope of an access control rule.
- It must be possible to exclude branches of the hierarchy from the scope of an access control rule.
- It must be possible to use static and/or dynamic groups to identify entries that should be included in or excluded from the scope of an access control rule. Groups will be discussed in more detail in a later section of this document.
- It may be worthwhile to consider the use of LDAP subtrees (as defined in RFC 3672) to identify entries that should be included in or excluded from the scope of an access control rule.

4.8. Delegation of Access Control Management

The access control policy for a Directory Server deployment is typically created and managed by server administrators with full control over the data contained in the server. However, in some cases it is desirable to allow other individuals to manage some portion of the way in which access control evaluation is enforced. This may manifest itself in a number of ways, including:

- In a directory containing multiple organizations, it may be desirable for an organizational administrator to define access control rules that apply to members of an organization. This could be very useful for products that allow for delegated administration like Sun Java System Access Manager and Messaging Server. In these cases, it must not be possible for the organizational administrator to see and/or modify any access control rules that exist outside of the scope of their own organization (unless they are explicitly authorized to do so), and they must also be prevented from defining access control rules that will interfere with the access granted to server-wide administrators.
- It may be useful to allow a manager to have limited control over what information is exposed about his or her direct reports. It may also be useful for this to have a cascading effect so that upper levels of management can override or delegate to lower levels of management.

- It may be useful to allow individuals to have limited control over what information is exposed about themselves, and potentially to whom that information is available.

If a directory environment makes use of the latter two cases, then there exists significant potential for a very large number of access control rules. This may be mitigated to an extent by storing some of this information outside of the access control rules. For example, it may be possible to define an access control rule that specifies which attribute(s) an individual may choose to expose or hide, and an attribute within the user's entry may be used to indicate which of those attributes the individual wishes to make available. A similar mechanism may be used in the entries for managers to control which attributes may be exposed regarding their subordinates. Note, however, that in that case there is the potential for access control evaluation to become expensive because of the potential need to retrieve multiple user entries in order to determine which attribute(s) to expose. It is likely the case that if this capacity provided it may require optimization in order to operate efficiently.

4.9. Access Control Synchronization

In the Sun Java System Directory Server, access control rules exist within the data that they are protecting, and they are also synchronized with it. While this is beneficial most of the time, that is not always the case. There are a number of exceptions to this, including:

- In the 6.0 release, a feature has been introduced in which it is possible to define access controls in the root DSE that apply to the entire server (these are called "global ACIs"). Because these access control rules do not exist in the data, they are not automatically synchronized and it is necessary for directory administrators to ensure that the same set of global ACIs are present in all instances.

- In some cases, it is not desirable to have the access control rules synchronized along with the data. For example, this is commonly the case for directory replicas that exist in sensitive areas (e.g., an extranet environment) in which only a subset of the data that is available in other server instances should be exposed. This may or may not be coupled with partial replication capabilities to ensure that the minimal set of information is synchronized, but it may still be beneficial to customize the set of access control rules for those instances.

- In some instances, data may exist in all directory instances but is not synchronized between them. For example, it may not be desirable to synchronize the Directory Server configuration, but it may still be necessary to control which users can access it.

- In some cases, different sets of access control rules should be applied to the same content. For example, in an environment with multiple data centers, there may be local administrators that should only have elevated access to the data contained in those local instances but not in instances deployed in other data centers.

The OpenDS access control model must provide a synchronization mechanism that is flexible enough to support these kinds of requirements. It should be possible to control which access control rules are synchronized and to which servers, and it should be implemented in a manner that allows the synchronization to be easily managed.

Note that there may be a number of complexities involved in the implementation of this functionality. In particular, if there are any dependencies between access control rules, then care must be taken to ensure that those rules are treated as a unit so that one does not appear without the others. Further, special attention must be paid to the case in which an access control rule is modified so that it is no longer in the scope of a given server, so that when it is removed from the excluded instances that it does not introduce any new security risks.

4.10. Precedence of Access Control Evaluation

In some cases, an access control policy may contain rules that conflict with each other. For example, one access control rule may allow a given operation while another may deny it. In these scenarios, the access control implementation must define an order of precedence that will consistently and predictably control which rule is enforced. This precedence should be clearly documented and easy to understand, yet powerful and flexible enough to meet the real-world needs of the directory deployment. It will likely not be sufficient to define absolute rules (e.g., "deny always takes precedence over allow"), but rather other factors will likely need to be taken into account (e.g., "rules with a target closer to the entry will have precedence over those with a target farther away from it, and in the event of a tie then deny takes precedence over allow").

4.11. Immediate Enforcement of Changes

In order to provide the highest level of security, any changes that occur in the server that may impact access control evaluation must take effect immediately. This includes changes to the access control policy itself (e.g., adding new rules, or removing or altering existing rules), but it must also include changes to other data that may impact access control evaluation. For example, if a user is added to or removed from a static group and there are access control rules that depends on membership within that group, then the evaluation of those rules must change accordingly. This should also apply to changes to a user entry that results in changes to dynamic group membership or other criteria used to select the set of users to which an access control rule applies.

As noted earlier in this section, changes to the access control policy are rare, and therefore it is acceptable to make recognition of such changes relatively expensive if it allows evaluation to be made faster. This may also be true for changes to a user entry that may alter the set of access controls that apply to that user, but wherever possible those expensive updates should be localized as much as possible. For example, if a user entry is updated, then it is likely the case that an expensive update will only be required for connections currently authenticated or authorized as that user.

4.12. Access Control and Virtual Attributes

Virtual attributes (not to be confused with data from virtual backends) are those attributes which don't actually exist anywhere in persistent storage but rather have their values computed at runtime. In the Sun Java System Directory Server, the primary use of virtual attributes is the Class of Service (CoS) facility. In OpenDS, we will provide similar capabilities, as well as a much richer interface for defining the behavior of virtual attributes in general. As of the initial draft of this document, the virtual attribute subsystem has not yet been implemented in OpenDS, but it is nevertheless important to consider the interaction between virtual attributes and access control.

There are two primary concerns regarding virtual attributes that must be addressed by the access control implementation:

- It must be possible to apply access control rules to determine whether a given virtual attribute value should be exposed to a client. In particular, it must be possible to control whether a user is allowed to see a given virtual attribute when returning an entry as part of search request processing, to determine whether a user is allowed to reference a given virtual attribute in a search filter, and whether a user is allowed to reference a given virtual attribute in the attribute value assertion of a compare request. It must also be possible to determine whether the client is allowed to set or remove one or more real value(s) for an attribute which might otherwise have its value(s) assigned by the virtual attribute subsystem.

- It should be possible to base access control criteria on the value(s) of virtual attributes. For example, if the `departmentNumber` for an individual is automatically set based on the value of the `manager` attribute in the user's entry, then it must be possible to use the value of the `departmentNumber` attribute to grant or revoke access to other data in the directory.

In both of these cases, the access control subsystem should essentially treat virtual attributes like real attributes, because the capabilities listed above are available for real attributes. The primary concern that should be addressed is therefore to ensure that all virtual attribute processing needed for an entry is completed before the access control processing. Note, however, that in order to achieve optimal performance it may be desirable to only compute virtual attributes if their values are actually needed for some reason. If the client has not requested that the virtual attribute value be returned and its value is not needed for any access control processing or for any other component within the server, then it should not need to be computed. It may therefore be useful to define a mechanism for other components in the server to inform the virtual attribute subsystem whether or not a given attribute will be needed for a particular operation.

4.13. Extensibility of the Access Control Implementation

When the access control subsystem of the Sun Java System Directory Server was initially designed, the requirements were relatively simple when compared with what is in use today, and even more so when compared with the requirements that are in place for OpenDS. It is therefore reasonable to assume that the access control requirements for OpenDS will change over time as new capabilities are added to the server. To address this, the OpenDS access control subsystem should be implemented in a manner that allows it to be easily extended to support new capabilities without introducing unnecessary impacts on performance or scalability and while still ensuring that it behaves correctly in all cases.

It may also be highly desirable to allow this extensibility to be exposed as a public API so that individual deployments may take advantage of this facility. It may very likely be the case that some environments will have access control requirements that could be met with some relatively simple logic but that cannot be achieved easily with the out-of-the-box access control facility that OpenDS provides. Exposing such a capability could provide a distinct advantage over other directory implementations, and this is a potential area in which the community could assist in improving and evolving the product to best suit their needs.

4.14. Alternate Expressions of Access Control Rules

As will be addressed in later section, the access control syntax used by the Sun Java System Directory Server is likely not the best representation for the native access control format used by OpenDS. These rules are not easily parsed, and there may be subtle differences in behavior based on seemingly equivalent ways of representing a given element. It is also not necessarily simple to construct new access control rules and add them into the server.

To address these concerns, it may be useful to consider supporting one or more alternate representations for access control rules. For example, one such alternate representation could be the syntax used by the Sun Java System Directory Server to serve as a migration path for existing deployments. However, it may also be useful to consider a format that can be more easily parsed and understood by clients. XACML is one possibility to consider, as would be the definition of our own expression.

4.15. Elimination of External Dependencies

Because the access control subsystem is a critical component of OpenDS, it should be designed without any external dependencies. We must have full control over the access control implementation and intimate knowledge of its inner workings. Basing access control determination in part on an external library reduces our ability to identify and correct any bugs that may exist and limits our flexibility to evolve the access control implementation in the future to meet customer demand.

4.16. Use as a General-Purpose Policy Engine

The primary focus for the OpenDS access control implementation should be protecting the data under the control of the Directory Server, as well as that data which may not be directly controlled by OpenDS but nevertheless may be exposed through it (as could be the case in proxied or virtualized access). At times, there have been thoughts of extending this capability to allow OpenDS to make access control decisions about data that is not under its control or exposed through it. While this may be an interesting possibility, it is not the primary role of OpenDS and is better suited for other products that already exist like the Sun Java System Access Manager and OpenSSO. After discussions with engineers responsible for that product, we do not believe that there is a significant need for OpenDS to serve in this capacity and that our efforts are better served ensuring that our access control subsystem provides the highest level of performance, scalability, and functionality for protecting directory data.

4.17. Disabling Access Control Evaluation

It is expected that the vast majority of directory deployments will have a need to make use of the access control facilities that OpenDS provides. However, there may be some cases in which this is not true. In particular, this includes instances in which OpenDS is embedded in some other product and not directly exposed to untrusted clients, and particularly those cases in which the performance of the directory solution is absolutely critical and the performance impact of an unneeded access control subsystem is undesirable. Under such conditions, it should be possible to completely disable the access control subsystem so that it has little or no impact on performance.

In practice, it will likely be the case that disabling access control will not actually eliminate that subsystem from the server, but rather will cause it to implement a default policy that will always allow any operation that may be requested. Such a default policy should be extremely efficient to implement without impacting the complexity of the core server implementation. This also introduces the possibility of other "built-in" policies that may be enforced at times, such as the "only allow root users connecting via the loopback interface" policy that was discussed earlier in this section. If a "canned" access control policy is acceptable for a given directory deployment, then it may be possible to enjoy excellent performance, ease of configuration, and adequate protection albeit at the expense of flexibility.

4.18. Community Approval and Review

Ultimately, the role of the OpenDS access control implementation is to allow directory deployments to provide the best level of protection possible for the data that they control and expose. Although the core OpenDS development team does have a history of significant customer interaction and knowledge of the problems and challenges that have been experienced with the Sun Java System Directory Server, it is important to ensure that the OpenDS implementation will be able to meet the needs of an even greater audience.

This document should be made publicly available for potential users to review and provide feedback so that we can ensure that we have addressed all requirements necessary to meet our intended goals. Further, when we are in the process of designing the native OpenDS access control implementation, that process should also be open for review and participation by the community, both in terms of comments and feedback, as well as potentially contributing code toward its implementation.

5. Areas for Improvement in Current Implementations

The access control subsystem in the Sun Java System Directory Server has a long and proven history of success as one of the most complete and feature-rich implementations available. Nevertheless, years of extensive use in diverse configurations have identified a number of elements that could be improved. This section will identify some of these areas so that the OpenDS access control implementation can be designed to avoid these issues.

5.1. The Access Control Rule Syntax

In the Sun Java System Directory Server, access control rules are stored in the `aci` attribute using a rich but somewhat complex syntax. The syntax used to express these rules presents several issues:

- The syntax is not necessarily intuitive and may take some time for administrators to learn. If we do choose a new syntax for OpenDS access control rules, we should make it much more user-friendly.

- It is not very convenient to make changes to an existing access control rule, particularly in entries containing multiple rules. We should try to make it easier to modify existing access control rules, and ideally should avoid the practice of allowing multiple rules to be stored in the same entry using multivalued attributes.

- There are often many ways to express the same logic, and they do not always have the same performance characteristics. The OpenDS access control syntax should be unambiguous and should attempt to limit the number of ways that a given rule may be expressed.

- In some cases, the order in which access control rules are listed in the multivalued `aci` attribute can have an impact on the performance of access control evaluation. OpenDS should attempt to internally order the rules that it evaluates in an attempt to avoid the most expensive rules if the access control decision can be made without them.

- Some elements of the access control policy are expressed using LDAP URLs, but this is a rather awkward encoding that is not always applicable. In particular, they never include a host and port specification, and in some cases the base DN element is not even a valid DN (e.g., "`ldap:///self`" or "`ldap:///anyone`"). OpenDS should provide an alternate mechanism for expressing this kind of information that is more appropriate and that does not require relaxing standards compliance when parsing elements.

Note that if we do decide to alter the access control syntax, we should do so in a way that will not interfere with the ability to migrate from the Sun Java System Directory Server. If OpenDS does adopt the ability to directly support the access control syntax of the Sun Java System Directory Server, then it will also likely be necessary to define an alternate syntax that is native to OpenDS and provides the ability to express elements not supported by the Sun Java System Directory Server.

5.2. The Meaning of "allow (all)"

Access control rules in the Sun Java System Directory Server must include an "`allow`" or "`deny`" section that indicates which rights are granted or revoked. It may be a comma-separated list of specific rights to provide (e.g., "`allow (read,search,compare)`"), but when used to grant rights to administrators it is more frequently "`allow (all)`". However, there are some notable drawbacks with this approach:

- In this context, "`all`" does not really mean all of the rights that are available. In particular, the "`proxy`" right is not included in the "`all`" set and there may be other cases in which permissions may not be suitable for inclusion in "`all`". This may not be immediately obvious to administrators and may lead to confusion about why administrative users aren't allowed to perform certain kinds of operations.

- The definition of "`all`" has changed over time. New capabilities have been added to it (e.g., "`import`" and "`export`") that were not present in earlier releases. This can potentially lead to security risks when upgrading the server because administrative users may be inadvertently granted rights that they were not expected to have. Ideally, upgrading the server should not impact the access control policy that is in place, however this will need to be balanced against the

- It is not possible to create user-defined profiles that combine various individual permissions together as a logical unit. If this were possible, then administrators would be able to customize the meaning of "`all`" to best suit their environment, and they could also create additional logical groupings (e.g., they may want to define something like "`readops`" to be the equivalent of "`read,search,compare`").

5.3. Considerations for targetattr Functionality

The Sun Java System Directory Server makes it possible to indicate the set of attributes to which access should be allowed or denied using the `targetattr` keyword. This is a list of attributes delimited with a double-pipe symbol (e.g., "`(targetattr="uid||givenName||sn||cn)")`"), it may also include an asterisk to indicate all attributes (e.g., "`(targetattr="*")`"), and it may include an exclamation mark to indicate that it applies to all attributes except the named set (e.g., `(targetattr!="userPassword")`").

While it is certainly a requirement to be able to target specific attributes in an access control rule, there are a few issues that have arisen around the use of the `targetattr` keyword which may not be obvious to administrators but can cause unexpected behavior in a directory environment. These include:

- For cases in all attributes are included (e.g., "`(targetattr="*")`"), and to an even greater extent for cases in which a specific set of attributes are excluded (e.g., "`(targetattr!="userPassword")`"), the set of target attributes will implicitly include all operational attributes as well as all user attributes. This has the potential to create rules that unintentionally allow users to do things that shouldn't be allowed by altering the values of certain operational attributes (e.g., those used for storing password policy state information or per-user resource limits). In OpenDS, it must not be possible to unintentionally grant access to operational attributes, and anything that is allowed should only come from rules that explicitly include those operational attributes.

- If there are multiple access control rules that use the "allow all except" form which have any users in common, then they may unexpectedly cancel each other out. For example, if one access control rule has a clause like "`(targetattr!="userPassword")`" and another has a clause like "`(targetattr!="uid")`" then the net effect will be to target all attributes because the first will implicitly allow access to uid and the second will implicitly allow access to userPassword. The OpenDS syntax should ideally be designed to prevent cases like this from occurring, and should attempt to warn administrators if inconsistencies like this are detected.

- Most of the time, the use of `targetattr` in a way that may have unexpected side effects (including both the "`(targetattr="*")`" and "`(targetattr!="userPassword")`" forms) comes from the lack of an easy way to specify a number of related attributes. OpenDS should provide mechanisms for targeting several attributes as a single logical group but in a way that could avoid these kinds of issues. For example, if the attributes are all in the same objectclass, then it may be useful to utilize the notation described in RFC 4529 (e.g., "`@inetOrgPerson`" to reference all attributes in the `inetOrgPerson` objectclass). However, if this is done, then it should be possible to exclude specific attributes from that objectclass (e.g., the equivalent of "all attributes in the `inetOrgPerson` objectclass except `userPassword`").

5.4. The Use of Wildcard DNs

The access control implementation of the Sun Java System Directory Server allows the use of wildcard characters in DN values to match any component. However, this implementation includes too much ambiguity around the meaning of the wildcard character. In cases with DN strings like "`uid=*,ou=People,dc=example,dc=com`" it is unclear whether the wildcard should match only a single attribute value. The obvious interpretation is that it will match a DN string like "`uid=john.doe,ou=People,dc=example,dc=com`", but it is possible that it could match multiple components of a single RDN (e.g., "`uid=john.doe+employeeNumber=5,ou=People,dc=example,dc=com`") or perhaps even multiple DN components, like "`uid=john.doe,ou=Central,ou=People,dc=example,dc=com`".

Another issue that will likely arise is that strings containing wildcard characters cannot be parsed as distinguished names. Therefore, it is not possible to use the existing DN normalization logic to make the comparisons more efficient.

If it is decided that the OpenDS access control implementation needs a facility like that provided by wildcard DNs then care should be taken to ensure that it can be interpreted efficiently and without ambiguity. This should also apply to similar capacities like macro ACIs, which use regular expression replacement values instead of wildcards.

5.5. Limitations of Proxied Authorization

The proxied authorization control provides a mechanism for one user to request that an operation to be performed under the authority of another. This is a very useful capability, and it is particularly helpful for cases in which an application maintains a pool of connections to the server and needs to perform various operations as a wide range of users. However, there are also a number of limitations that prevent it from being used as widely as it could. These include:

- The only information included in the control value is the authorization ID for the user as whom to perform the operation. This does not provide any mechanism to account for things other than the user identity that may be taken into account during access control processing (e.g., the real address of the client system, whether the client is using a secure communication channel, the mechanism that the client used to authenticate to the target application, etc.). It would be helpful to be able to include other information that may prove useful in the course of access control evaluation.

- The proxied authorization control does not provide any mechanism for verifying the identity of the target user. The Directory Server is required to trust that the application requesting the operation on behalf of that user has performed sufficient validation of the user's identity. It would be helpful to include a mechanism for allowing the client to include some mechanism for verifying the identity of the end user.

- The proxied authorization control is only useful in cases in which the target Directory Server instance is able to resolve the authorization ID to a user entry. This is not suitable for cases in which the Directory Server does not know anything about the target user, which could be the case if remote data is involved, as in distributed and virtual environments. It would be helpful to be able to include additional information about the user and the capabilities that they should be extended (much like a SAML assertion is able to provide in federation environments).

- The proxied authorization control cannot be used in conjunction with bind operations. For applications which need to authenticate users and use pooled connections, then it is often necessary to maintain a separate pool for processing bind operations. It would be helpful to have a mechanism for determining whether a particular bind operation would be successful without actually changing the identity of the authenticated user. Note that this capability may not be possible for all forms of SASL authentication, as it may not be possible to include the credentials needed to process the bind (e.g., as would be the case when using SASL EXTERNAL to authenticate based on an SSL client certificate).

It may be very useful for OpenDS to define an enhanced proxied authorization control that could be used as an alternative to the standard version described in RFC 4370 and includes additional functionality to address these limitations.

6. The Questions that Must Be Answered

In school, students are frequently taught that there are six important questions that should be answered when investigating or reporting on a problem:

- Who?
- What?
- When?
- Where?
- Why?
- How?

This is an appropriate set of questions to apply in the design of the access control subsystem. The question of "Why?" (e.g., "why does user X need access to data Y and/or need to be denied access to Z?") will be left to administrators to answer, but the access control implementation must provide the capability to implement the desired policy no matter what the reasons. The remaining five questions can be used as a guideline to ensure that the implementation will meet at least a minimal set of needs.

6.1. Who?

This question may be expanded to "Who should be allowed (or should not be allowed) to perform a given set of operations?". This is clearly an important question that is one of the foundations of access control. It must be possible for access control rules to specify "Who?" in any of the following ways:

- The DNs of a specific set of users
- The DNs of groups of users
- Dynamic selection criteria (e.g., all users in this branch that match this filter)
- The user whose entry is being targeted (i.e., "self")
- A relationship between the user and the target (e.g., the target user's manager)
- The set of all authenticated users
- The set of anonymous/unauthenticated users
- The set of any user, whether authenticated or not

Clearly an important factor in answering the question of "Who?" rests in the identity of the user. As such, the access control implementation must provide a way to define constraints in which the identity has been established. This may include the type of authentication (e.g., was it simple or SASL authentication, and if it was SASL then which mechanism was used) as well as additional information about whether there is an authorization in place that is different from the identity of the authenticated user (e.g., through the use of the proxied authorization control or a SASL authorization ID).

6.2. What?

The question of "What?" can be expressed in two parts. The first is "What will the user be allowed (or not allowed) to do?". In order to be able to answer this question it will be necessary to understand the entire set of things that can be done in the server. This list may be expanded over time (and the access control implementation must allow for that), but it includes:

- The ability to establish a connection to the server.
- The set of operations that may be included, including abandon, add, bind, compare, delete, extended, modify, modify DN, search, and unbind operations.
- The ability to include certain kinds of controls in the request
- The ability to see certain information in the responses provided
- The ability to perform administrative tasks like LDIF import/export, database backup/restore, server shutdown/restart, etc.

The set of operations for which it must be possible to allow or restrict access will be discussed in detail in subsequent sections.

The second question is "What data may be targeted by those operations?". It must be possible to scope an access control rule so that it can apply to one or more entries as described in the Access Control Scoping requirement mentioned earlier in this document. Further, it must be possible to narrow the scope even further to specify which attributes may be targeted, and even more specifically to a given value or set of values.

6.3. When, Where, and How?

Although it is possible to break down these into individual questions, they can all be placed under the umbrella of "Under what circumstances will users be allowed (or not allowed) to perform a requested operation?". It must be possible to take any of the following elements into account when making access control decisions:

- The time that the operation was requested. This should be able to take into account the time of day, the day of the week, the day of the month (potentially including fuzzy expressions like "the first Monday"), and the month of the year.

- The address of the client. This should be able to take into account either the IP address (IPv4 or IPv6) and the resolvable host or domain name for the client. In the event that the client request has passed through multiple hops (e.g., through a Directory Proxy Server instance), then it should ideally be possible to base the decision on the actual client address rather than the address of the last hop.

- The protocol that the client is using to communicate with the server. In the event that the client request has passed through multiple hops (e.g., through a DSML to LDAP gateway) then it should ideally be possible to address the entire communication channel.

- Whether any security measures are in place for the client communication. This should be able to take into account integrity checking (i.e., whether it is possible to ensure that communication is not altered in transit) and confidentiality (i.e., whether it is possible to ensure that communication is not observed in transit). It may also be interesting to consider whether the client request has been signed as per RFC 2649.

- The set of controls included in the request. At a minimum, it should be possible to restrict which controls may be used based on their OID, but for specific controls it may be useful to consider additional constraints (e.g., which attributes may be requested in the read-entry or matched values controls).

7. Root Users and the Privilege Subsystem

The Sun Java System Directory Server offers a single root user (typically "`cn=Directory Manager`"), which is the most powerful account in the directory. It is never subject to access control checking and can do things that other accounts cannot. In Directory Server 6.0, changes have been made to reduce the set of things that only the root user account could do, but there still exist a number of things that can't be done by anyone else. Unfortunately, there are some aspects of this capability that leave room for improvement. In particular:

- There is only a single root user account, which must be shared by whoever needs it. This creates auditing concerns because it is not necessarily possible to tell who actually used the account.
- The root user doesn't actually have an entry anywhere, but rather is just comprised of a DN and a password. This means that it is only possible to use LDAP simple authentication to bind as that user, and potentially stronger SASL mechanisms are not available for this user.
- Some information, and in particular information that may be useful in helping to debug problems (e.g., tombstone entries) will only be made available to the root user and will not be exposed to any other user regardless of the access control configuration.
- Because there is only a single root user account that must be shared, changing the password for the root user can be difficult because the change must be coordinated between all appropriate users and the new password must be securely communicated to only those people that need it. In some cases, applications may also use the root account for the work that they perform (although this is not advisable, it is sometimes done because of the performance penalties that may be associated with the access control implementation), which even further adds to the complexity of root password changes.
- It is not possible to subject the root user to any form of password policy, nor is it possible to apply any restrictions to root user authentication (e.g., to ensure that it is only allowed to authenticate from a specific set of client systems, or that it is only allowed to operate over a secure communication channel).
- If a client has a valid need to be able to perform some operation that only the root user can do, then the client must be given full root access, which may grant that client other capabilities that they should not have.

These are all issues that must be addressed within OpenDS. Some of them are outside the scope of access control, but they have been mentioned here for the sake of completeness. In particular, the following are already true of root user accounts in OpenDS and have been implemented outside of the access control subsystem:

- It is possible to have any number of root users in OpenDS by creating the entries below `cn=Root DNs,cn=config` and including the `ds-cfg-root-dn` objectclass in the entry. Each authorized administrator may have his or her own account with separate credentials, and whenever an administrator leaves or changes positions, that account may be removed without impacting any of the other administrators.

- It is possible to enforce password policy constraints for root users. This can include features like password expiration, validation for new passwords, and the requirement to use secure authentication and/or secure password changes.

- Each root user account has its own entry in the server, and root users have the ability to use SASL authentication mechanisms.

7.1. Privileges in OpenDS

At the present time with no access control subsystem in place, there is essentially nothing that distinguishes root users from any other user (including unauthenticated users). This is intentional, because there must not be any operation within OpenDS that will require a root user. That is, it must be possible to create non-root users with appropriate capabilities to perform any action within the server. This should be done by identifying a number of different kinds of tasks that administrators may need to perform and making them privileges that can be assigned to individual users.

An excellent model to use for this mechanism is the process rights management (also called least privilege) capability found in Solaris 10. This facility makes it possible for a normal user to do things that would otherwise only be allowed for root users (e.g., create a process that listens on a privileged port). It consists of a number of individual privileges that may be granted to or revoked from a user, as well as named privilege sets that can be used to easily reference the privileges that they encapsulate (e.g., the "basic" privilege set includes the privileges that will be extended to normal users by default, although it is possible to grant additional privileges and/or revoke any of these basic privileges from a user).

In OpenDS, the privileges that we define may include:

- The ability to bypass access control evaluation
- The ability to modify the server's access control configuration
- The ability to read the server configuration
- The ability to change the server configuration
- The ability to perform read operations using JMX
- The ability to perform write operations using JMX
- The ability to subscribe to JMX notifications

- The ability to invoke LDIF import/export operations
- The ability to invoke backend backup/restore operations
- The ability to invoke a server shutdown or restart
- The ability to use the proxied authorization control
- The ability to terminate arbitrary client connections
- The ability to cancel arbitrary client requests
- The ability to request unindexed searches
- The ability to reset user passwords
- The ability to participate in a data synchronization environment

Note that in general the use of privileges should not be dependent upon the access control subsystem. Rather, the code to determine whether to allow a given operation based on the privileges assigned to a user will likely be either in the core server or in the component used to perform the specified operation. However, in some cases it may be that both the access control subsystem and the privileges subsystem will both be consulted when performing an operation (e.g., when creating a task to perform an LDIF import, the access control subsystem will verify that the user can add the task entry to the server while the privilege subsystem will verify that the user has the ability to perform the LDIF export).

7.2. Scoped Privileges

Although the ability to assign individual privileges may be very useful and can eliminate cases in which users are given root access in order to be able to perform only one tasks, there are cases in which an individual privilege may not be as fine-grained as might be desired. For example, in a directory environment hosting information for a number of organizations, it may be useful to define organizational administrators that have elevated privileges just within that organization. For example, it would be extremely helpful to allow a user to bypass evaluation within a given organization, but any operation outside of that range would be subject to access control checking. The types of scoping that may be appropriate vary between privileges, but may include:

- Restrict access control bypass to a specific subtree
- Restrict unindexed searches to a specific subtree
- Restrict password reset authorization to a specific subtree

There may be other restrictions that are appropriate, but in some of those cases access controls may be sufficient to enforce them. For example, when granting access to the server configuration it may be useful to restrict that access to a specific set of configuration entries, but it should be possible to enforce that restriction through the access control subsystem so there is no need for a scoping mechanism to provide that capability.

7.3. Privilege Sets

OpenDS should provide the ability to create named privilege sets that can be assigned to users rather than the individual privileges that they contain. It will be necessary to create at least one privilege set for root users that encapsulates the following privileges

- The ability to bypass access control evaluation
- The ability to modify the server's access control configuration
- The ability to read the server configuration
- The ability to change the server configuration
- The ability to invoke LDIF import/export operations
- The ability to invoke backend backup/restore operations
- The ability to invoke a server shutdown or restart
- The ability to terminate arbitrary client connections
- The ability to cancel arbitrary client requests
- The ability to request unindexed searches
- The ability to reset user passwords

Root users should inherit this privilege set by default, and it may be possible to assign this privilege set to individual users to make them equivalent to root users.

It must be possible to create custom privilege sets, and to modify the privileges included in a privilege set. In addition, when assigning a privilege set to a user, it must be possible to grant additional privileges not contained in that set and to explicitly revoke privileges contained in that set.

7.4. RBAC-Like Authorizations

Solaris has long provided support for role-based access control (RBAC), which provides a couple of interesting features, including:

- The ability to define roles in the system. Roles are essentially identical to user accounts, except that they cannot directly authenticate but may only be assumed by authorized users.

- The ability to grant a user or role the capacity to execute a particular command under the authority of another user (including the root user).

These concepts could also be very helpful when applied to access control in OpenDS. In particular, it could be useful to define user accounts with elevated privileges that do not have the ability to directly authenticate to the server but that may only be accessed when authenticated as an authorized user. It is possible that the proxied authorization control could be used to achieve this capability, but it is likely not sufficient on its own because no additional credentials are required.

Ideally, it should be possible for a user to request that an operation be performed under the authorization of a privileged user. It must be possible to restrict the set of users who can assume that authorization, and it must also be possible to require additional credentials in order to assume that authorization. One way to achieve this could be to restrict the ability to authenticate as certain users based on the authentication or authorization identity of the client prior to beginning of the subsequent bind process as described in the access control requirements for bind operations. It may also be possible to achieve this kind of functionality through an enhanced proxied authorization control.

8. Group Management Requirements

Groups are a vital component of an access control implementation because they make it possible to easily apply a uniform set of access control rules to a specific set of users. While it should be possible to define an access control policy that does not reference any groups, it will often be much more convenient to define groups for this purpose. This section describes the capabilities that OpenDS should provide in the area of group management, particularly as it is relevant to access control management. There are other uses for groups in the Directory Server, but they are outside the scope of this document.

8.1. Legacy Static and Dynamic Groups

Historically, the term "group" applied to an LDAP directory server refers to an entry with the `groupOfNames` or `groupOfUniqueNames` objectclass. These are also called static groups because they contain explicit lists of their members (in entries with the `groupOfNames` objectclass, the DNs of the members are contained in the `member` attribute, and in entries with the `groupOfUniqueNames` objectclass, the member DNs are contained in the `uniqueMember` attribute). Static groups are well supported among directory servers and LDAP-enabled applications, but they have inherent scalability problems because performance degrades as the number of entries increases. There may also be problems with nesting in static groups, since some applications expect to be able to list the DN of another group in the `member` or `uniqueMember` attribute and have the members of that group considered part of the larger parent group. This practice is potentially very expensive for the server to detect and handle.

Dynamic groups attempt to address many of the management problems associated with static groups by defining criteria for membership rather than an explicit list of members. This is accomplished through the `groupOfURLs` objectclass, with the `memberURL` attribute containing an LDAP URL that provides the base, scope, and filter that comprise the membership criteria. Attempting to determine whether a given user is a member of a large dynamic group will almost certainly be faster than making the same determination for an equally-sized static group.

OpenDS must support the use of traditional static groups and dynamic groups for the purposes of access control evaluation. It should be possible to configure whether to attempt to resolve nested static groups, with this capability disabled by default for performance reasons (the enhanced group definition described below should provide a suitable alternative for group nesting). If nesting is allowed, then it may also be useful to define a maximum depth that should be searched when following nested group references.

8.2. Enhanced Group Definitions

As discussed above, static groups have problems with scalability and nesting. Dynamic groups generally address these problems, but they introduce problems of their own. In particular, it is not possible to explicitly add or remove users from a dynamic group, nor is there any mechanism for nesting of any kind. OpenDS should create an enhanced group object that can address these shortcomings and combine the benefits of static and dynamic groups without many of their drawbacks.

An enhanced group must provide the following capabilities:

- It must provide the ability to define a set of criteria that may be used to dynamically identify group members.
- It must provide the ability to explicitly include individual users as members based on their DNs, even if they do not match the membership criteria.
- It must provide the ability to explicitly exclude individual users as members based on their DNs, even if they match the membership criteria.
- It must provide the ability to include nested groups based on their DNs. The nested groups may be enhanced groups, as well as standard static or dynamic groups.

It is recommended that enhanced groups be based on an auxiliary objectclass so that an existing static or dynamic group may be converted to an enhanced group simply by adding this objectclass. The dynamic membership criteria should be based on the `memberURL` attribute and static member inclusion should be allowed using either the `member` or `uniqueMember` attribute. Custom attributes will be required for identifying the DNs of excluded members and nested groups.

Note that there are similarities between this enhanced standard group object and the object proposed in the LDAP: Dynamic Groups for LDAPv3 Internet Draft proposed by Novell. However, this proposal is not sufficient for OpenDS for a number of reasons:

- The Novell proposal does not include any provision for nested groups.
- The Novell proposal suggests the use of the `memberQueryURL` attribute for dynamic membership criteria rather than `memberURL`. This would make it incompatible with legacy dynamic groups which use `memberURL` to define this criteria.
- The Novell proposal does not include OIDs for any of the custom schema elements that it defines.
- The Novell proposal never received any significant traction in the LDAP community, nor is it in widespread use. There is therefore no significant reason for us to consider supporting that draft instead of or in addition to our own proposal.

8.3. The memberOf Virtual Attribute

OpenDS must provide at least one virtual attribute that includes the DNs of the groups in which a user is a member. The Sun Java System Directory Server 6.0 provides a similar capability through the `isMemberOf` attribute, although this attribute only includes the DNs of the static groups in which the user is a member. There is no analog that includes dynamic group membership, although role membership may be determined through the `nsRole` attribute.

The OpenDS implementation must define a virtual attribute that lists the DNs of all groups in which a user is a member. This must include all static, dynamic, and enhanced groups. If there is significant demand for variants of this virtual attribute that list only static (or only dynamic or only enhanced) group memberships, then we can consider providing additional virtual attributes with that capability. It must be possible for the administrator to specify the name of the attribute in which this group membership information will be published, although for reasons of backward compatibility the default name should be "`isMemberOf`".

Note that while "`memberOf`" may be a more logical choice and is in use for this purpose in other directory implementations, OpenDS must not use this attribute for this purpose in the default configuration because it may create incompatibilities with other products. In particular, the Sun Java System Access Manager already makes use of a non-virtual attribute named `memberOf` for its own functionality. However, it must be possible for an administrator to configure this virtual attribute to be named "`memberOf`" if they have no intention of using any product that expects to use that attribute in a conflicting manner.

8.4. Virtual Static Groups

Despite their clear benefits over static groups in terms of management and scalability, dynamic groups have not been widely adopted in directory deployments. A common reason for this is that the applications customers wish to use with their servers do not include support for dynamic groups. Even though the Sun Java System Directory Server has the ability to use dynamic groups internally for features like access control, in many cases these same groups are also used for other reasons by directory clients and therefore a "least common denominator" approach must be used. It is almost certainly the case that the introduction of enhanced group support will encounter the same barrier to adoption.

In order to address these concerns, OpenDS must provide the ability to construct virtual static groups. A virtual static group is an entry that appears to be a standard static group (either based on the `groupOfNames` or `groupOfUniqueNames` objectclass) but whose membership is dynamically determined by referencing a dynamic or enhanced group. Virtual static groups must exhibit the following characteristics:

- Any search operation that does not reference the membership attribute (either `member` or `uniqueMember`) in the filter and does not return this attribute to the client should not require any special processing.

- Any search operation that references the membership attribute in the filter but does not return this attribute to the client should perform only the minimum amount of processing required to return the result. It must not be necessary to construct the entire member list solely for the purpose of determining whether a given user is a member.

- Any search operation that does not reference the membership attribute in the filter but does return this attribute must return the complete member list.

- For any search operation that references the membership attribute in the search filter and also returns this attribute to the client, it must be possible for the administrator to configure whether to return the complete member list or a minimized member list containing only member DNs referenced in the search filter.

- Any compare operation that references the membership attribute must perform the minimum amount of processing required to return the result.

8.5. Group Management Operations

Despite the relatively widespread use of groups in LDAP directory servers, client interaction with groups is often incorrect or inefficient. Further, most directory implementations include their own non-standard grouping technology and it is unreasonable to expect clients to understand all of the possible implementations and how best to interact with them. For these reasons, OpenDS should define facilities for standardizing the way that clients may interact with grouping mechanisms in directory servers. This should include the following:

- A control which may be included in a search request that may be used to determine whether a given user is a member of a specified group or set of groups. It should be included in search operations targeted at one or more group entries, and only groups in which the specified user is a member should be returned to the client.

- A control which may be included in a search request targeted at a user entry to determine the set of groups in which that user is a member. The filter and attribute list contained in the search request should be applied to the groups in which the target user is a member, which may be used to further restrict the set of results returned.

- A control which may be included in a search request targeted at a group entry to retrieve the entire set of members for that group. The filter and attribute list contained in the search request should be applied to the entries of the members for that group to further restrict the set of results returned.

- An extended operation which may be used in an attempt to add a user to a group. Note that this capability may not be available for all types of groups.

- An extended operation which may be used in an attempt to remove a user from a group. Note that this capability may not be available for all types of groups.

At present, there are no known RFCs, Internet Drafts, or other proposals to define these types of mechanisms. It would be very beneficial for us to create one or more Internet Drafts for this topic so that other implementations could adopt this functionality. The more directory implementations that provide support for these mechanisms, the more benefits that clients will receive from them.

8.6. Performance Versus Accuracy

In order to achieve a high degree of performance from our access control subsystem, group interactions must be very efficient. It is likely that some form of caching may be employed to achieve the best performance possible, but in the course of doing so there must be some consideration for reacting to membership changes in a timely manner. In particular, if a user is granted some capability by virtue of membership in a specified group, then that capability must be revoked immediately if a change occurs that removes the user from that group. There must not be any requirement for a user to disconnect and reconnect in order to be subject to those changes. Conversely, although it is less critical from a security perspective to ensure that new capabilities granted as a result of adding a user to a group take effect immediately and with no additional administrative interaction, this is highly desirable from a usability perspective.

The OpenDS access control subsystem must recognize and apply the following group-related changes immediately:

- Adding or removing/excluding one or more members in a static or enhanced group (including through nesting relationships).
- Adding or removing one or more nested groups in an enhanced group (or in a static group with this capability enabled).
- Altering the membership criteria for a dynamic or enhanced group.
- Altering a user entry in a manner that impacts dynamic or enhanced group membership.
- Creating a new group that is referenced by one or more access control rules.
- Removing an existing group that is referenced by one or more access control rules.

8.7. Maintaining Referential Integrity

The Sun Java System Directory Server includes a referential integrity plugin that may be used to remove or update references to a user whenever that user's entry is deleted or renamed. OpenDS must provide a similar facility with the following characteristics:

- It must be possible to completely disable this capability
- We should determine whether it is appropriate or desirable to have it enabled by default
- It must be possible to customize the attributes for which to maintain referential integrity
- The referential integrity subsystem should ensure that all configured attributes have a DN syntax
- The referential integrity subsystem should make an effort to ensure that all configured attributes are properly indexed
- It must be possible for referential integrity work to be processed synchronously (i.e., before the delete or modify DN response is sent to the client) or asynchronously
- When operating in an asynchronous manner, it may be useful to build the candidate list synchronously but perform the associated cleanup asynchronously
- Special consideration may be required to ensure proper operation in a synchronized environment

In addition to these requirements, special consideration should be given to the interaction between referential integrity and access control. In particular, it must be decided how to handle delete and modify DN operations targeted at entries referenced in access control rules. Such changes have the potential to be dangerous, and it will likely be the case that the most appropriate action is to alert administrators of the situation and allow them to manually apply any appropriate changes to the access control configuration.

8.8. Non-Standard Group Membership Behavior

Static groups based on the `groupOfNames` and `groupOfUniqueNames` objectclasses are defined in the LDAPv3 specifications, in RFC 2256 and more recently in RFC 4519. In both cases, the `groupOfNames` objectclass requires that the `member` attribute be present, and the `groupOfUniqueNames` objectclass requires that the `uniqueMember` attribute be present. That is, neither type of static group should be allowed to exist without any members.

The OpenDS definition of these objectclasses deviates from the standard to allow either type of group to exist with no members. This decision was made for a number of reasons:

- The Sun Java System Directory Server also defines these objectclasses with the membership attribute optional rather than required. Using the same configuration in OpenDS promotes backward compatibility and makes the migration path easier.

- Making the membership attribute required significantly increases the complexity of group management for the case in which only one member is left in the group. If this member is to be removed, then it may not necessarily be desirable to remove the entire group entry, particularly if there are references to it (e.g., in access control rules). It is also the case that clients may have permission to alter group membership but do not have sufficient permission to delete a group entry.

- Making the membership attribute required also creates a significant problem for the referential integrity mechanism when attempting to delete a user that happens to be the last member of a group. It may not be desirable to automatically delete the group in this scenario, nor would it be desirable to leave a reference to a nonexistent user in the membership list (especially if there is a chance that another user will be created with the same DN but who should not be considered a member of that group).

- Making the membership attribute required introduces the potential for race conditions in directory environments. If an attempt to remove a user from a group is rejected because it was the last member, then the client could respond by attempting to delete that group. However, it is possible that another client could have added another member in the process, and therefore the group need not (and should not) be removed. A variant of this race condition is also possible in a synchronized environment if a client adds one member to a group and then removes another, which could occur if the changes are made through separate modifications that get routed to different servers (e.g., by an intermediate proxy).

It is very unlikely that deviation from the standard definitions of these group objectclasses will cause any problems for any clients. In order for a problem to occur as a result of this change, it would be necessary to have a group with no members and one or more client applications that do not work properly under these conditions. In this case, the problem could be resolved by simply removing the group with no members or ensuring that it has at least one member.

8.9. The Deprecation of Roles

The release of the iPlanet Directory Server 5.0 introduced a new capability called roles, which is still present in the Sun Java System Directory Server (including the upcoming 6.0 release). Roles are conceptually similar to groups, but they are a proprietary feature not implemented in any other server and therefore not widely used by clients. This is at least partially guaranteed by the use of Netscape-specific naming in the implementation (e.g., using schema elements like `nsRole`, `nsRoleDN`, `nsManagedRole`, `nsFilteredRole`, etc.). They offer very little benefit over the use of more standard grouping mechanisms in the Sun Java System Directory Server, and are completely redundant in OpenDS. As a result, OpenDS will not include direct support for this non-standard feature.

In many cases, the removal of role support will not have any impact at all because they are not currently in widespread use. For cases in which roles are used, it is a relatively simple matter to convert them to groups, and the mechanism used to migrate existing installations supporting roles should perform this translation automatically. If there are any clients that have specific dependencies on the proprietary roles implementation, then it should be possible to address these concerns through attribute aliasing and/or virtual attributes.

In the process of removing support for roles, OpenDS must continue to provide compatibility with legacy clients for some period of time. It is strongly recommended that any application which depends on this proprietary roles implementation be redesigned so that it uses a more standard mechanism based on standard static static or dynamic groups, or on enhanced groups for which we expect to draft a proposal for standardization as described later in this document.

9. Access Control Requirements for Connection Acceptance

The Sun Java System Directory Server 6.0 will introduce a new feature that makes it possible to reject client connections based on various criteria (primarily based on the address of the client). OpenDS has expanded on this capability, not only making it possible to define sets of allowed or denied clients, but also to create post-connect plugins that can apply much more flexible logic in deciding whether to allow communication with a particular client.

Neither the implementation in the Sun Java System Directory Server nor the current implementation in OpenDS make the ability to reject a client connection part of the access control subsystem. However, it may be appropriate to alter the OpenDS implementation so that it is part of the access control mechanism. This would provide the following benefits:

- It would make it easier for the access control implementation to enforce the fail-safe degradation mechanism discussed earlier in this document, in which the server must only allow connections over the loopback interface and will only allow root users to authenticate in the event that a problem is detected with the access control configuration.
- It would remove the requirement for each connection handler to individually define its own mechanism for determining whether to allow a connection from a given client. It must still be possible to define different criteria for different connection handlers, but that logic does not necessarily need to be part of the connection handler implementation.

OpenDS must provide a mechanism for rejecting client connections based on the address of the client. Whether that mechanism is absorbed into the access control implementation is to be determined, but it is absolutely necessary for the access control subsystem to be able to reject clients in the event that an access control inconsistency is detected.

It may also be worthwhile to consider the possibility that the OpenDS access control implementation could provide the ability to terminate client connections under appropriate conditions. For example, it may be useful to provide the ability to terminate a client connection if an attempt is made to authenticate as a root user with invalid credentials or using an insecure mechanism or from a system that is not allowed to bind as that user. This and similar options are discussed in greater detail in a later section.

10. Access Control Requirements for Operation Processing

The primary focus for the access control mechanism in any directory server should be to determine whether the server should allow a requested operation. There are a number of generic requirements for access control that are applicable to any type of operation (e.g., the time of day, the address of the client, whether the communication is secure, how the user authenticated, controls included in the request, etc.), but these have been addressed in other sections of this document. This section attempts to address the access control requirements that apply to specific kinds of operations.

10.1. Abandon Operations

The abandon operation may be used to request that the server stop processing on an operation that had been previously requested. It is only available for use on the same connection that was used to request the original operation and therefore there are few security concerns around the use of abandon operations. The Sun Java System Directory Server does not define any mechanism for enforcing access control for abandon operations.

However, there may be a case in which additional access control checking may be appropriate. In particular, it may be desirable to check whether the authorization DN for the abandon request is equal to the authorization DN for the operation being abandoned. If the authorization DNs are the same, then the operation should be allowed without any further checks. However, if they are different then the user associated with the authorization DN for the abandon request should have the privilege that allows canceling arbitrary client requests.

This access control check may help prevent cases in which one user may interfere with operations requested by another when asynchronous operations using the proxied authorization control are allowed on the same client connection. Although such applications should be rare, it is at least theoretically possible for a malicious user to repeatedly generate abandon requests with various target message IDs.

The server must also ensure that clients are not allowed to perform any unexpected processing as part of the abandon request through the inclusion of unauthorized controls in the request.

There is no response for an abandon operation, so if an abandon request is refused by the access control subsystem (or is rejected for any other reason), then the abandon request will simply have no effect.

10.2. Add Operations

Add operations are used to create new entries in the Directory Server. An add request consists of an entry DN and a set of attributes. The OpenDS access control implementation must provide the ability to take any combination of the following elements into account when processing an add operation:

- The target location for the entry.
- The RDN attribute(s) for the entry.
- The structural objectclass for the entry.
- The presence or absence of any auxiliary objectclasses for the entry.
- The presence or absence of any specific attribute types in the entry.
- The presence or absence of any specific attribute values in the entry.
- Whether the entry matches a given search filter

Note that while it may be possible to introduce restrictions on some of these elements through other means (e.g., the allowed RDN attribute(s) may be controlled through a combination of DIT structure rules and name forms), the ability to do so through the access control subsystem will likely provide greater flexibility and ease of use.

10.3. Bind Operations

Bind operations are used to authenticate to the Directory Server, or in the case of an anonymous bind request to eliminate any previous authentication that had been performed. There are three components within any bind request:

- The protocol version. For LDAP requests, the version should be either 2 or 3. For other protocols, version numbering may differ or may not be appropriate.

- The DN of the user that is authenticating. This is primarily used only in the case of simple authentication, as most SASL mechanisms provide an alternate means of identifying the target user.

- The credentials for use in the authentication. This includes a tag that identifies whether the bind request uses simple or SASL authentication. For simple authentication, it should also include the password. For SASL authentication, it should include the name of the SASL mechanism and an optional set of credentials.

The access control subsystem in the Sun Java System Directory Server does not deal with bind operations in any way, so it is not possible to define any restrictions for bind operations in that implementation. This is a significant shortcoming, and the OpenDS access control mechanism must provide the ability to enforce access control evaluation for bind operations. It must be possible to take any combination of the following elements into account:

- The protocol version from the bind request
- The authentication DN from the bind request
- The authorization DN from the bind request, if there is one (this may be available for certain SASL mechanisms)
- The authentication type (simple or SASL) from the bind request
- The SASL mechanism from the bind request
- The authentication and/or authorization DN held by the client at the beginning of the bind request processing (before being eliminated at the beginning of the bind operation processing)
- Any attribute type and/or attribute value from the user's entry
- Whether the user's entry matches a given search filter
- Any groups in which the target user is a member

Note that there are a number of interesting elements that must be considered when providing an access control implementation for bind operations. One such issue lies in the mechanism used to determine the identity of the user that is authenticating to the server. For simple authentication, this is typically the bind DN contained in the request, although there may be cases in which that is not accurate (e.g., if no password is provided then it must be treated as an anonymous bind regardless of whether there is a bind DN, and in some cases the requested DN may be mapped to some other value, like "`cn=Directory Manager`" could be mapped to "`cn=Directory Manager,cn=Root DNs,cn=config`"). For SASL authentication, the identity of the user that is authenticating is not available until the authentication is largely complete, and may take multiple request-response cycles to verify. In general, this means that access control processing for bind operations should occur in two stages: early on in the processing, a check should be made whether the client is allowed to attempt authentication with the specified protocol version and mechanism, and then any validation based on the identity of the user must be made at the end of the process.

Another issue that may arise when performing access control processing for bind operations is that SASL binds may require multiple request-response cycles. In particular, the CRAM-MD5 and DIGEST-MD5 mechanisms each require two stages of processing, while GSSAPI requires three. Therefore, any access control processing which relies on the identify of the user must only occur after all stages have completed. Further, if there is any processing that relies on the identity of the client before the bind processing started, then that information must be retained across all intermediate stages of the bind request processing. In order to remain extensible, the access control implementation should not attempt to perform any processing or interpretation of the SASL credentials. In order to determine whether the SASL bind is still in an intermediate stage, it should rely solely on the result code for the bind operation (a result of SASL_BIND_IN_PROGRESS indicates an intermediate stage, while SUCCESS indicates that the bind has completed; any other result code indicates a failed bind).

It is important to note that access control for bind operations should be enforced for all users, regardless of the target identity. This is in part based on the constraint that the identity of the user may not be known until late in the process, but even if that were not the case it would still be highly desirable to ensure that bind-based access controls were always evaluated for all authentication attempts. This makes it possible to restrict the conditions under which root users may authenticate to the server, for example to ensure that they are only allowed to bind from certain client addresses and/or with a specific authentication mechanism. No such restrictions are currently available in the Sun Java System Directory Server, and this will be a welcome enhancement in a number of environments.

Conversely, it is also important to note that there is little or no value in attempting to craft access control rules that will prevent anonymous authentication attempts. For security reasons, any current authentication state must be cleared when initiating a bind operation, and the client connection will be left in this state if the bind attempt should fail for any reason. Since this is exactly the same state that would be used for the client connection if the anonymous bind had succeeded (and this is also the state for the client connection before any authentication is performed), then there is no point in attempting to reject it. The only effective way to prevent an anonymous bind attempt would be to terminate the client connection, but this is undesirable because there are valid cases in which it may be useful to perform an anonymous bind (e.g., to clear any current authentication state for a client connection that is to remain established for potential re-use in handling subsequent operations). The only case in which it makes any sense at all to reject an anonymous bind request is the case in which the bind request contains a valid DN but no password. According to the LDAPv3 specification, this is valid but should result in an anonymous bind, although this is responsible for a number of security problems in poorly-designed clients. OpenDS has a mechanism for rejecting these bind requests outside of the access control mechanism, although there are cases in which it may be useful to handle this in the access control subsystem (e.g., for cases in which it needs to be allowed for one particular type of application, perhaps based on the client address, but should be rejected for all other clients).

Another interesting capability that enforcing access controls for bind operations may provide is the ability to create user accounts that operate like Solaris roles in that it is not possible to authenticate directly to that account but rather only authorized users may assume them. This may be exposed through the mechanism that provides information about the previous authentication held by the client connection before bind processing began. If the access control implementation provides a means of only allowing authentication as user X if the client was previously authenticated as user Y, then this could provide a mechanism for multi-factor authentication or step-up authorization while preserving a clear audit trail.

10.4. Compare Operations

Compare operations may be used to determine whether a particular entry contains a specified attribute value. There are three components that make up a compare request: the DN of the target entry, the attribute type, and the assertion value. It must be possible to make access control decisions for compare operations based on any combination of the following elements:

- The DN of the target entry
- The attribute type from the compare request
- The assertion value from the compare request

- Any attribute type and/or attribute value from the target entry
- Whether the target entry matches a given search filter
- Any group membership for the target entry

10.5. Delete Operations

Delete operations may be used to remove entries from the Directory Server. The delete request contains only the DN of the entry to be deleted. It must be possible to make access control decisions for delete operations based on any combination of the following elements:

- The DN of the target entry
- Any attribute type and/or attribute value from the target entry
- Whether the target entry matches a given search filter
- Any group membership for the target entry

10.6. Extended Operations

Extended operations may be used to perform processing for operations not defined in the core LDAP specifications. Extended requests contain two elements: an OID that identifies the type of request to be processed, and an optional value that contains information to use when processing that request.

Extended operations currently supported by OpenDS include:

- StartTLS -- This may be used to initiate secure communication over an insecure channel. It allows what is essentially the equivalent of SSL communication over a port typically used for non-SSL communication.

- Who am I? -- This may be used to allow the client to request the authorization ID based on the current authentication state.

- Password Modify -- This may be used for user password changes or administrative password reset operations. It includes a mechanism for providing the user's current password for validation (which makes it suitable for users to be able to change their own passwords after they have expired), and potentially for generating a new password.

- Cancel -- This may be used to cancel an operation that had been previously requested on the same connection. It is similar to the abandon operation, but ensures that the server will provide responses for both the cancel request and the target operation.

However, due to the inherent extensibility of these types of operations, it is possible to create additional extended operations that may be used for virtually any purpose. Because of this flexibility, it is not possible for the access control implementation to understand all types of extended operations that may be defined, which can pose a challenge when it comes to deciding whether or not to allow a given operation. The access control implementation must provide a mechanism for making decisions about whether to allow an extended operation based on its request OID. In addition, any internal operations performed during the course of processing the extended operation (e.g., updating the user's password in the course of processing the password modify extended operation) should also be subject to access control. Access control for internal operations is discussed later in this section.

It is permissible for the access control implementation to include special support for specific types of extended operations that enhance the protection that can be offered for those operations. However if such support is added, it must not interfere with the ability to enforce access control for extended operations for which there is no form of enhanced support.

10.7. Modify Operations

Modify operations may be used to alter the contents of entries in the Directory Server. A modify request contains two elements: the DN of the entry to be modified and a set of one or more attribute-level modifications. An attribute-level modification includes the attribute modification type (add, delete, replace, or increment), the attribute type, and an optional set of attribute values.

It must be possible to make access control decisions for modify operations based on any combination of the following elements:

- The DN of the target entry
- Any of the attribute types and/or values included in the modification
- Any of the attribute modification types
- Any attribute type and/or attribute value contained in the entry before the modification
- Any attribute type and/or attribute value contained in the entry after the modification
- Whether the target entry matches a given search filter before the modification
- Whether the target entry matches a given search filter after the modification
- Whether the modify operation attempted to alter the structural objectclass for the entry
- Whether the modify operation attempted to alter any of the auxiliary objectclasses for the entry

10.8. Modify DN Operations

Modify DN operations may be used to change the DN of an entry, potentially including the DNs of its children (also called a subtree rename), and potentially including moving the entry below a new parent. The modify DN request may contain four elements:

- The current DN of the target entry

- The new RDN for the entry
- A flag that indicates whether to delete the old RDN attribute value(s) from the entry
- An optional new parent DN for the entry

It must be possible to make access control decisions for modify DN operations based on any combination of the following elements:

- The DN of the target entry
- Whether the target entry has any children
- Whether the target entry will be moved below a new parent
- The DN of the new parent for the target entry
- The new RDN attribute(s) and value(s) for the entry
- Whether the old RDN attribute value(s) are to be removed
- Whether the operation will alter the RDN attribute(s) for the entry
- Whether the target entry matches a given search filter before the modify DN operation
- Whether the target entry matches a given search filter after the modify DN operation
- Any attribute type and/or attribute value from the target entry
- The structural objectclass for the target entry
- Any auxiliary objectclasses contained in the target entry

10.9. Search Operations

Search operations may be used to retrieve entries contained in the Directory Server based on a provided set of search criteria. The search request contains the following elements:

- The base DN for the search
- The scope for the search (baseObject, singleLevel, wholeSubtree, subordinateSubtree)
- The alias dereferencing policy (neverDerefAliases, derefInSearching, derefFindingBaseObj, derefAlways)
- The maximum number of entries that should be returned from the search (the size limit)
- The maximum length of time in seconds to spend processing the search (the time limit)
- A flag that indicates whether to return both attribute types and values or just attribute types
- The filter to identify matching entries
- The set of attributes to include in matching entries returned to the client

It must be possible to make access control decisions for search operations based on any combination of the following elements:

- The base DN from the search request
- The scope from the search request
- The alias dereferencing policy from the search request
- The size limit from the search request
- The time limit from the search request

- The value of the typesOnly flag from the search request
- Any attribute type and/or attribute value contained in the search filter
- Any matching rule ID referenced in extensible match filter components
- Any of the requested attributes
- Any attribute type and/or attribute value contained in the search base entry

Search operations are interesting from an access control perspective because it is not only necessary to decide whether to allow the search operation to be processed, but also to protect any data that may be returned in the course of processing the request. This may include both search result entries (i.e., entries which match the provided search criteria) and search result references (i.e., referrals to other directories in which the search may be processed).

For each search result entry, the access control mechanism must be able to prevent that entry from being returned if the client should not have any access to see that entry, and if the entry is to be returned then it must be possible for the access control implementation to remove entire attributes and/or specific attribute values if the client is not authorized to access that information. All access control decisions must be made against the complete entry as it is available internally within the server (including any virtual attributes that should be returned to the client or that may need to be involved in access control determinations), and not against the contents of the entry after it has been pared down based on the requested attribute list (which may not contain sufficient information to make accurate access control decisions). In addition to the criteria already noted for the search request itself, it must be possible to take any combination of the following elements into account when deciding whether to allow and/or how to sanitize a search result entry:

- The DN of the entry to return
- The depth of the entry relative to the search base DN
- Any attribute type and/or attribute value contained in the entry
- Whether the entry matches a given search filter
- Any groups in which the target entry is referenced

Search result references include one or more LDAP URLs referencing other locations that may have entries matching the search criteria. It must be possible to prevent a search result reference from being returned, or to remove one or URLs from the reference list. In addition to the criteria already noted for the search request itself, it must be possible to take any combination of the following elements into account when deciding whether to allow and/or how to sanitize a search result reference:

- The address of the target server from the referral URL
- The port to use when communicating with the target server from the referral URL
- The base DN for the target server from the referral URL

10.10.Unbind Operations

The unbind operation is used to indicate that the client wishes to terminate the connection with the server. The client should close the connection immediately after sending the request to the server (and if it doesn't close the connection, then the server will).

Since the server cannot prevent the client from disconnecting, there is little benefit to providing any form of access control for unbind operations, although the access control subsystem must ensure that the client is not able to perform any unexpected processing by including unauthorized controls in the unbind request. If the access control implementation decides to prevent unbind request processing for any reason, then it must terminate the client connection.

10.11.Internal Operations

As the name implies, internal operations are those operations that have been requested from within the server rather than by an external client. Internal operations may be initiated through a number of components, including plugins, identity mappers, extended operation handlers, backends, control processing, and other elements within the server. It is almost certainly the case that the access control subsystem will also be required to perform internal operations in order to obtain all of the information required to make the necessary decisions.

It must be possible to perform internal operations without any kind of access control processing for cases in which this processing is not performed on behalf of an external client, or for cases in which the developer is certain that no access control will be required. However, it must also be possible to ensure that access controls are enforced for internal operations that are the direct result of client requests. For example, if a client attempts to use the password modify extended operation, then the access control mechanism must ensure that the client has the ability to request this operation based on the OID of the extended request, but the server must also verify that the internal modify operation used behind the scenes to actually change the user password is also allowed for the client.

11. Access Control Requirements for Request Controls

Like extended operations, controls make it possible for the Directory Server to support additional capabilities that were not explicitly included in the protocol specification. Note, however, that because controls must be handled in the course of processing an operation, the server is not able to allow as much extensibility in adding support for new controls as can be provided for extended operations.

At a minimum, the Directory Server must provide the ability to allow (or not allow) the use of a given control based on its OID. This will ensure that it can support new controls as soon as they are added, without the need for special support for that control in the access control implementation. Further, since there are some limited capabilities for components to define their own supported controls, this does provide a way to allow the server to properly handle them as well.

There are a number of cases in which the access control subsystem should provide specific support for particular controls. The remainder of this section describes those controls and the access control capabilities that should be provided when using them. Note that as support for additional controls are added to OpenDS, special consideration must be given to whether any access control enhancements will be required for those controls.

11.1. The LDAP Assertion Control

The LDAP assertion control may be used to ensure that the associated operation is only processed if a specified assertion filter matches the target entry for the operation. The OpenDS access control implementation must provide a mechanism to ensure that the client has permission to perform a base-level search against the target entry using the provided assertion filter.

11.2. The LDAP Pre-Read and Post-Read Controls

The LDAP pre-read and post-read controls may be used to retrieve an entry as it appeared either immediately before or immediately after processing for a particular operation. The OpenDS access control implementation must provide a mechanism to ensure that the client has permission to read the target entry and to see all requested attributes. If the client does not have permission to read the target entry, then the control processing must not be allowed. If it is determined that the client has permission to read the target entry, then the access control mechanism must remove any attribute values from the entry that the client does not have permission to see.

11.3. The Server-Side Sort Control

The server-side sort control may be used to request that the server sort the results before they are returned to the client. The OpenDS access control implementation must ensure that the client has permission to issue a search request containing inequality filters based on the attribute types specified in the sort criteria.

11.4. The Proxied Authorization V1 and V2 Controls

The proxied authorization controls (both version 1 and version 2) provide the ability to request that an operation be processed under the authority of another user. The OpenDS access control implementation must provide a mechanism to ensure that the authenticated user is able to perform operations as the target user.

12. Access Control Requirements for Administrative Tasks

Most directory clients will only need to access the "user data" contained in the server. However, it is also possible to request administrative operations over protocol, and these actions will need to be protected as well. These operations are generally performed through the tasks interface, which requires entries to be created in the `cn=tasks` backend. While it is possible to prevent normal users from being able to schedule tasks simply by not allowing them to create entries below `cn=tasks`, if an administrator needs to be able to perform one or more of these functions then it is not necessarily as convenient to use the same mechanism to control which particular tasks can be scheduled.

A previous section already discussed the use of privileges to indicate whether a user should be allowed to perform various administrative operations. This is important, but it may also be very helpful to have special access control support for these operations in order to make it possible to add additional restrictions on how they may be used.

Access control options that we may wish to consider for administrative operations include:

- For LDIF import operations, it should be possible to make access control determinations based on the backend into which the data is to be imported, whether to replace existing data or append to it, and whether it should be possible to skip schema validation.

- For LDIF export operations, it should be possible to make access control determinations based on the backend from which the data is to be exported.

- For backup operations, it should be possible to make access control determinations based on the backend(s) to be archived.

- For restore operations, it should be possible to make access control determinations based on the backend to be restored.

- For index creation and regeneration operations, it should be possible to make access control determinations based on the target backend, the attribute type(s), and the index type(s) to generate. It should also be possible to take into account whether it is a new index or an existing index.

- For index verification operations, it should be possible to make access control determinations based on the backend, the attribute type(s), and whether to iterate through id2entry or a particular index.

- For server shutdown and restart operations, it should be possible to make access control determinations based on whether it is a shutdown or restart and whether a shutdown reason has been provided.

12.1. Offline Administrative Operations

The primary focus when enforcing access control for administrative operations must be on processing those operations through the OpenDS task interface. Any administrative action taken using offline tools (i.e., those that do not communicate with a running server over protocol) fall outside the scope of this access control mechanism.

In such cases, it may be possible to introduce an authentication mechanism for those tools so that they will only be allowed for root users defined in the configuration, but in that case there will be a number of limitations (e.g., it would likely be restricted to password-based authentication). At any rate, if such a capability is to be added at some point then it would likely not interact with the OpenDS access control implementation and therefore is outside the scope of this document.

The only other remaining consideration in this area would be that of authentication for offline administrative operations that are requested from an external interface, particularly one that is invoked from a remote system. If this capability is available, then there must be some facility (e.g., an agent) for executing commands on the remote system. This will absolutely require some form of authentication and authorization in order to perform such actions. However, this will likely be a part of the administrative interface used to initiate the operation and will not necessarily have any integration with the OpenDS access control subsystem.

13. Auditing and Debugging

Certainly the most important requirement for the OpenDS access control implementation is that it must work correctly. Whenever a request is made, the server must evaluate that request properly against the configured access control policy and determine whether to allow it to be processed. This requirement also includes a number of corollaries:

- It must be possible to determine what rights a given user has, particularly in the context of a given operation.
- If a problem occurs that causes the access control subsystem to return an unexpected result, then it must be possible to quickly debug the issue and determine its underlying cause.
- It must be possible for the server to collect and report information in a manner that will be useful for auditing (i.e., determining what has been done by whom and under what conditions).
- It should be possible for the server to perform special processing based on various events that occur during access control processing.

This section will provide additional detail on each of these capabilities.

13.1. General Access Control Debugging

The OpenDS access control implementation must include facilities for auditing and debugging the processing that it performs. This implementation must be designed so that it has little or no performance impact when it is not in use (and ideally minimal performance impact when it is in use), but it must also operate in a manner that makes it possible to obtain this information in a targeted manner to yield the most relevant results.

The Sun Java System Directory Server provides a log level that may be used for debugging the operation of the access control subsystem. Although it may be useful for debugging problems in a test environment, it is generally unsuited for use in production servers because it offers very little flexibility. If this facility is enabled, then the server will write debug information to its error log for every access control decision that is made. In a busy production environment, this will generate a great deal of debug data, and it may be very difficult to identify information relevant to a specific problem because of the large amount of information about other unrelated operations that are in progress at the same time.

Although OpenDS should provide the ability to enable full access control debugging if that is really desired, it will likely be the case that a more focused mechanism will be more helpful in real-world environments. Therefore, it must be possible to enable access control debugging on a per-connection or potentially per-operation basis. This should include the following:

- A control that can be included in an individual request to enable debug information for just that operation.
- An extended operation that can be used to request that debugging be enabled for subsequent operations on the current client connection, or for an arbitrary client connection in the server. There should also be a corresponding extended operation that may be used to end debugging on that connection. It must not be necessary to send the "start debugging" and "stop debugging" requests on the same client connection.
- An extended operation that can be used to define criteria for automatically enabling debugging for future connections that have not yet been established. This should at least include the ability to base the determination on the client address and the DN used to authenticate to the server. There should also be a corresponding extended operation that may be used to cancel this auto-debug mechanism. It must not be necessary to send the "start debugging" and "stop debugging" requests on the same client connection.

It is likely the case that these debugging capabilities will not be specific to access control but rather will be part of a larger debugging mechanism for the entire server. Other components within the server will need similar capabilities, and it will be useful to take advantage of that rather than implement something specifically for access control.

13.2. The GetEffectiveRights Control

The Sun Java System Directory Server provides support for a GetEffectiveRights control which may be used to determine the capabilities that a specified user may have when interacting with a given entry in the server. The behavior of this control is described in the documentation for the Sun Java System Directory Server at http://docs.sun.com/source/817-7613/aci.html#wp20922. OpenDS must provide support for this control, as it is used by some LDAP client applications in order to programmatically determine the operations that a user may perform on a given entry.

Note that the version of the GetEffectiveRights control supported by the Sun Java System Directory Server is different from the specification included in section 9 of the Access Control Model for LDAPv3 Internet Draft. For backwards compatibility, the version of the GetEffectiveRights control implemented in the Sun Java System Directory Server must be preserved. There is little or no need to consider supporting the version of this control defined in the draft, as we do not intend to support the access control model defined in that specification and it does not provide any additional capabilities not already available in the version of the control supported in the Sun Java System Directory Server.

13.3. The Get Applicable ACIs Control

The GetEffectiveRights control described above provides a mechanism for determining what will be allowed when interacting with a given set of information in the Directory Server. However it does not necessarily provide all of the information which might be useful for debugging access control issues. Another capability which might be useful in this context is the ability to retrieve the set of access control rules that are applicable to a particular operation. Whereas the GetEffectiveRights control will allow an administrator to determine what is allowed (or not allowed), a "get applicable ACIs" control would provide a mechanism for determining which access control rules are evaluated in the process of making this decision.

It must be possible to combine this control with the LDAP No-Op control in order to determine the set of access controls that would be evaluated for an operation without actually attempting to process that request. It should also be possible to use this control without combining it with the LDAP No-Op control in order to determine the set of access controls that were evaluated when processing that operation. In either case, the set of evaluated access control rules should be returned in the response regardless of whether the operation was successful, provided that it reached the point of access control processing.

When the get applicable ACIs control is used in conjunction with the LDAP No-Op control, it should be possible for an authorized user to embed additional information in the control that may override properties that may otherwise be derived from the client connection. This will allow administrators to make determinations on behalf of other users and clients. Information that should be possible to override in this manner includes:

- The IP address and/or resolvable name for the client
- The authentication and/or authorization identity
- The mechanism used to authenticate to the server
- The type of security (if any) used for the communication between the client and the server
- The time and date at which the operation has been requested

Note that no specification exists for such a control, and we are therefore free to design it in any way which best meets our needs. However, it should be designed in an extensible manner so that it can evolve over time to address changing needs and further access control development. This could also be an excellent candidate for proposing as a standard extension to the protocol, as will be discussed in a subsequent section.

13.4. Access Control Auditing Capabilities

OpenDS must provide the ability for administrators to determine exactly what information was accessed or altered and by whom for every operation processed in the server for the purpose of maintaining an audit trail. For performance reasons, it should be possible to customize the set of information that is captured and/or to filter the set of operations for which auditing is performed, since the auditing requirements will vary between environments and reporting unnecessary information will impact both performance and storage consumption. Ideally, it should be possible to configure the server to report any combination of the following:

- For any operation, the address of the client, communication protocol, authorization identity, connection ID, operation ID, request message ID, request controls, response controls, result code, response message, matched DN, and referral URLs.
- For abandon operations, the target message ID, the target operation ID (if available), and the result of the abandon attempt.
- For add operations, the DN and attributes of the entry to be added.
- For simple bind operations, the protocol version, bind DN, and authenticated DN.
- For SASL bind operations, the SASL mechanism and authenticated DN.
- For compare operations, the DN of the target entry and the attribute type contained in the assertion.
- For delete operations, the DN of the target entry.
- For all extended operations, the OID of the extended request and the OID of the extended response.
- For cancel extended operations, the target message ID and the target operation ID (if available).
- For password modify extended operations, the DN of the target user, a flag indicating whether the current password was provided, and a flag indicating whether a new password was provided.
- For "Who Am I?" extended operations, the returned authorization ID.
- For modify operations, the DN of the target entry and the set of modifications.
- For modify DN operations, the DN of the target entry, the new RDN, a flag indicating whether to delete the old RDN value, and the new superior DN (if applicable).
- For search operations, the base DN, scope, dereferencing policy, size limit, time limit, typesOnly flag, filter, and requested attributes. For each search result entry, the DN and set of attributes returned. For each search result reference, the set of referral URLs.
- For unbind operations, no additional information should be needed.

Note that it is likely that this need could be met by the server's access logging framework. The current implementation is not suitable for this purpose, but we do intend to redesign it so that it is better suited to meeting the needs of the server. These requirements for access control auditing should be taken into account in the course of designing the new access logging mechanism.

13.5. Events Triggered by Access Control Processing

The auditing capabilities mentioned above would be useful for "after the fact" analysis of events that happen in the server, but the server should also provide a mechanism for defining actions that should be taken at the time that certain access control decisions are made. Candidates for this type of processing could include logging the operation, generating an administrative alert, or terminating the client connection. This could provide a mechanism similar to targets in the Linux netfilter/iptables implementation or obligations in the XACML specification.

13.6. Security Considerations for Access Control Debugging

The access control debugging capabilities in OpenDS must not compromise security in any way. The following points must be considered when designing and implementing the debugging framework:

- Any debugging facility that may be enabled or disabled over protocol must itself be subject to access control evaluation and only made available to authorized users.

- It must be possible to sign and/or encrypt any audit information that may be written to disk.

- The debugging facility should make every attempt to avoid exposing particularly sensitive information like user passwords. If passwords are to be included in audit messages (e.g., because they are included in the set of attributes for an add request or the set of changes for a modify request), then they should be used in encoded form if possible. It may be useful to have an additional configuration option for enabling debugging for potentially sensitive data.

- Special consideration should be given for the behavior of the server under conditions in which audit information cannot be written (e.g., the disk to which log information is written becomes full). Some administrators may wish to have the ability to shut down the server or have it stop accepting requests if it enters this kind of state.

14. Access Control Administration

There are a number of feature requirements for managing aspects of access control in OpenDS. In all cases, the features listed in this section should be exposed both through graphical interfaces (e.g., Web applications, Swing / Java Web Start fat clients, etc.) and command-line utilities. The features associated with access control administration include:

- The administrative interfaces must operate completely within the bounds of access control enforcement. They must never expose any information to users that they are not allowed to see, nor perform administrative changes that they are not allowed to make, nor invoke administrative tasks that they are not allowed to execute. It should be possible for users to have read-only access to some elements of the configuration.

- All administrative operations must be performed with an authorization identity of the authenticated authenticated user. This will ensure that the audit trail will be properly preserved for all administrative operations performed through this interface, and will ensure that no user is allowed to perform any operation that would not otherwise be allowed through any other interface.

- The administrative interfaces must provide a mechanism for enabling and disabling access control evaluation in the server. It must also be possible to transition the server between the normal mode of operation and the mode in which only root users will be allowed to submit requests over the loopback interface.

- The administrative interfaces must provide mechanisms for viewing, creating, deleting, and editing access control rules. They should provide warnings for any change that the server detects may have unintended consequences (e.g., removing a rule that others depend on, or multiple rules with "allow all except").

- It must be possible for users to view, add, delete, and edit access control rules over protocol without the use of any special administrative tools. It must be possible to represent access control rules as individual elements in LDIF so that they are easily visible (e.g., the format must not include anything that would require any part of the rule to be base64-encoded) and may be individually edited (e.g.., it must not be necessary to retrieve and edit a "blob" in order to add or remove one or more access control rules, and modifying an access control rule should not require interacting with any other rules). It must be possible for an administrator to manage the access control rules defined in the server using only simple tools like ldapsearch and ldapmodify if they so desire.

- The administrative interfaces should provide a "sanity check" mechanism that can look at the entire policy and determine whether there may be any potential problems or inconsistencies with any of the access control rules.

- The administrative interfaces must provide a mechanism for viewing, creating, managing, and removing root user accounts.

- The administrative interfaces must provide a mechanism for viewing and editing the individual privileges and privilege sets for a given user, including root and non-root users. The administrative interfaces must also provide a mechanism for viewing, creating, deleting, and editing privilege sets.

- The administrative interfaces must provide a mechanism for viewing, creating, deleting, and editing group definitions, including static groups, dynamic groups, and enhanced groups.

- The administrative interfaces must provide mechanisms to determine whether a given user is a member of a specified group, to determine the set of all groups for a given user, and to determine the set of all members for a specified group.

- The administrative interfaces must provide a mechanism for displaying the effective rights for a given user when interacting with a specified entry.

- The administrative interfaces must provide a mechanism for displaying the set of applicable ACIs that would be involved in a given operation, without actually performing that operation.

- The administrative interfaces must provide a mechanism for enabling and disabling targeted debugging facilities. It must be possible to turn access control debugging on or off for a given client connection, and it must be possible to create, delete, and modify criteria that will be used to automatically select new client connections for which debugging will be enabled.

- The administrative interfaces must provide a mechanism for viewing access control debug information. They must also provide a mechanism for viewing audit data, as well as enabling basic analysis for that data (e.g., break-down of recent operations by type, requester, result code, etc.).

15. Considerations for Standardization

In order to implement all functionality defined in this requirements document, it will be necessary to develop a number of custom extensions. Some of these extensions are for the benefit of clients, and it will be necessary for them to know how to utilize them to gain their full advantage. We can build support for such extensions into any client applications and utilities that we provide, and we can also provide APIs for using them in custom applications. However, in order to get the greatest degree of adoption for these capabilities, it may be best to propose them for standardization (e.g., in the form of Internet Drafts).

OpenDS extensions to consider as proposed standards include:

- The schema elements (objectclasses and their associated attribute types) and behavior of enhanced groups.

- The group management controls which may be used to determine whether a given user is a member of a specified group, the set of groups in which a given user is a member, and the set of members for a specified group.

- The group management extended operations which may be used in an attempt to add a given user to a specified group, or to remove a given user from a specified group.

- The enhanced proxied authorization control which may be used to address many of the limitations of the current proxied authorization control.

- The get applicable ACIs control which may be used to request the set of access control rules that apply to a particular operation.

16. Common Development and Distribution License, Version 1.0

Unless otherwise noted, all components of the OpenDS Directory Server, including all source, configuration, and documentation, are released under the Common Development and Distribution License (CDDL), Version 1.0. The full text of that license is as follows:

```
1. Definitions.

    1.1. "Contributor" means each individual or entity that creates
         or contributes to the creation of Modifications.

    1.2. "Contributor Version" means the combination of the Original
         Software, prior Modifications used by a Contributor (if any),
         and the Modifications made by that particular Contributor.

    1.3. "Covered Software" means (a) the Original Software, or (b)
         Modifications, or (c) the combination of files containing
         Original Software with files containing Modifications, in
         each case including portions thereof.

    1.4. "Executable" means the Covered Software in any form other
         than Source Code.

    1.5. "Initial Developer" means the individual or entity that first
         makes Original Software available under this License.

    1.6. "Larger Work" means a work which combines Covered Software or
         portions thereof with code not governed by the terms of this
         License.

    1.7. "License" means this document.

    1.8. "Licensable" means having the right to grant, to the maximum
         extent possible, whether at the time of the initial grant or
         subsequently acquired, any and all of the rights conveyed
         herein.

    1.9. "Modifications" means the Source Code and Executable form of
         any of the following:

        A. Any file that results from an addition to, deletion from or
           modification of the contents of a file containing Original
           Software or previous Modifications;

        B. Any new file that contains any part of the Original
           Software or previous Modifications; or

        C. Any new file that is contributed or otherwise made
           available under the terms of this License.

    1.10. "Original Software" means the Source Code and Executable
          form of computer software code that is originally released
          under this License.

    1.11. "Patent Claims" means any patent claim(s), now owned or
```

hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. "Source Code" means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or

INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as "Participant") alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48