Abstract

Authentication, authorization, and administration are critical components in secure internet computing. This paper presents a unified architecture for provisioning these services in cloud and enterprise environments. The architecture aims at providing robustness, performance, scalability, high availability, and low total cost of ownership (TCO) for deployments, while preserving the compatibility with existing IT infrastructure. The design scales vertically and horizontally by distributing workload among multiple servers. The flexible cache configuration minimizes network access traffic, delivering the high performance demanded by peak business workloads. The architecture offers high availability via replication services.

The design is the result of the cumulative efforts of the open source community in the past many years, involving technologies such as Name Service Switch (NSS), Pluggable Authentication Modules (PAM), Lightweight Directory Access Protocol (LDAP), LDAP Content Synchronization, Secure Socket Layer/Transport Layer Security (SSL/TLS), OpenLDAP, Proxy Cache, nss-pam-ldapd, and so on.

Keywords: Cloud Computing, Enterprise Computing, High Availability, Scalability, Security and Trust, E-commerce, Authentication, Authorization, LDAP
Introduction

This paper presents a unified architecture for provisioning authentication, authorization, and administration (hereafter AAA) services in cloud and enterprise computing environments. The design adopts standardized Pluggable Authentication Modules (PAM) [1] and name services application programming interfaces (APIs) that offer compatibility with existing IT infrastructure. The design further integrates with the OpenLDAP SLAPD Directory servers [2] for performance, scalability, and high availability. Multiple OpenLDAP servers can be deployed to distribute and share workloads. The OpenLDAP Proxy Cache [3] configuration caches information locally, which minimizes network access traffic. The LDAP [4] Sync Replication (syncrepl) services [2] are used for replication and high availability.

Cloud computing providers can deploy virtual machines preconfigured with an AAA module as virtual appliances on which additional software stacks can be developed, deployed, and redeployed. Virtual appliances can be commissioned and decommissioned based on dynamic business needs.

Enterprises can deploy services in a modular and scalable manner based on corporate geographical regions and organizational units. One or more servers can be deployed for each geographical region to minimize cross-region access latency. Clients can further be configured to access services for their respective organizational units from the regional servers. The cache configuration on the client systems stores information locally, eliminating redundant accesses to the regional but remote servers. This is particularly critical for heavily loaded systems in which repeated service requests for the same information can be handled via local cache.

Currently the unified AAA architecture is available on Linux and Solaris systems in the open source community. The supported version is branded as Symas Unified User Management version 4, or SUUM v4 for short.
1. Evolution of Related Technologies

The unified AAA architecture design evolved from Name Service Switch (NSS) and Pluggable Authentication Modules (PAM) [1] to PADL’s nss ldap and pam ldap [5], to nss-ldapd [6] and to the OpenLDAP Name Service Overlay (nssov) [2]. The nssov overlay enhanced nss-ldapd with PAM support. The code was contributed back to nss-ldapd, turning nss-ldapd into nss-pam-ldapd [6]. The OpenLDAP project further extends the architecture with various technologies, such as the Proxy Cache extension [3] and the LDAP Sync Replication (syncrepl) engine [2].

Name Service Switch (NSS) and Pluggable Authentication Modules (PAM)

Traditionally Unix-like systems require various name services in order for the systems’ AAA to function properly. The name service information in system local storage such as /etc/passwd is accessible via the system libc library [7]. The system local storage using flat files is constrained in terms of performance and scalability. Furthermore, flat files cannot be readily shared by multiple systems, even when those systems share the same configuration.

Additional name service modules, such as Network Information Services (NIS/NIS+), were developed to support alternative database stores and facilitate data sharing. The configuration file (/etc/nsswitch.conf) further specifies the order in which databases should be consulted for name service information, e.g.

```
passwd: files nis
shadow: files
group: files nis
```

PAM [1] is a pluggable authentication framework in which multiple modules can be stacked together to provide authentication services based on the needs of each individual application. The framework further divides functionality into four different types of modules which can be implemented and plugged in separately:

- Authentication - for user identity and password verification
- Account Management – for access control checking, such as whether a user can access an account or a particular service, whether an account has expired, and so on
- Session Management – for maintaining session-related information, e.g., for later billing services, for compliance and auditing
- Password – for password-related operations, e.g., for password changes

**PADL nss_ldap and pam_ldap**

Directories have become popular in an enterprise IT infrastructure in which enterprise information can be stored centrally and hierarchically in a distributed database. Luke Howard of PADL [5] introduced two libraries, nss_ldap and pam_ldap, that integrate LDAP directories into the aforementioned name services and pluggable authentication framework. The two libraries provide a set of C library extensions that translate system AAA calls into LDAP requests, thereby enabling the storage of security information in distributed and scalable LDAP directories, Figure 1.

![Figure 1: PADL nss_ldap and pam_ldap](image-url)
The PADL approach was a big step forward and has become popular in many large deployments. As the approach gained popularity, a number of opportunities for improvement also surfaced:

- **Symbol pollution** - In most deployments nss_ldap is loaded at run time by the system *libc* library into an application’s address space. If nss_ldap is built as a shared library, it will also import its dependent libraries. The symbols the libraries bring with them can clash with those of similar libraries that the application needs. For example, applications, such as *ssld*, which are linked with their own SSL/TLS [8] libraries, begin to crash when nss_ldap is added to a system. One solution is to use static linking for nss_ldap, which eliminates the symbol pollution problem, but leads to the issue that follows.

- **Bloated library** - The nss_ldap library itself contains logic to translate between name service calls and LDAP operations. When the SSL, LDAP, and other dependent libraries are statically linked in, the total size can grow up to over one megabyte. While most modern operating systems can share library text segments (i.e., the code portion) between processes, there is a correspondingly large data segment needed by each instance of nss_ldap. The data segments cannot be shared between processes.

- **Non-reentrant** - Many parts of nss_ldap are non-reentrant because they make use of libraries that are not themselves reentrant. These non-reentrant parts of nss_ldap must be protected with mutexes, which will negatively impact performance of multi-threaded applications.

- **Chatty** - nss_ldap makes at least one query for each of the name service calls. If a site uses groups with many members, certain calls, such as the *initgroups* call, may result in a couple of thousand LDAP queries. The overhead of performing the ASN.1 [9] encoding, SSL/TLS encryption, TCP/IP exchange, and subsequent SSL/TLS decryption and ASN.1 decoding adds to the processing and time overhead. Depending on network latency and LDAP server performance, an initgroups call may take 30 to 45 seconds to complete.
• Poor to non-existent caching - nss_ldap itself performs no caching. If a process repeatedly requests the same information, nss_ldap will perform an LDAP query each time. Using nss_ldap with a name service caching daemon (nscd) only improves matters slightly, as it caches single entries, but no cross-entry associations such as for group membership. In addition, the nscd daemon supports only name services, but not PAM.

• No connection sharing - nss_ldap opens one connection to an LDAP server for each process that is running and making use of name service calls. On a busy system, this can mean a few hundred connections to an LDAP server. On most LDAP servers, this will result in the available file descriptors being exhausted in fairly short order.

• No disconnected operation - In most applications, nss_ldap requires a constant connection to its LDAP server. If an LDAP server becomes unavailable, name service stops working.

• Poor performance over high latency or low-bandwidth networks

The nss-pam-ldapd NSLCD Daemon

The nss-pam-ldapd NSLCD daemon developed by Arthur de Jong [6] also provides LDAP support for system’s AAA services, Figure 2. The design includes nss_ldap and pam_ldap as the C library extensions similar to the PADL’s approach, except that those extensions do not issue LDAP client calls directly to LDAP servers. Instead, the library extensions communicate with the nss-pam-ldapd NSLCD daemon via the NSLCD protocol through a Unix domain socket. The approach modularized LDAP-related functionalities into a separate process, which addressed many issues in the original PADL design, including library version and dependency control, threaded vs. non-threaded issues, library initialization and shut-down issues, and so on.
2. Unified AAA architecture using OpenLDAP SLAPD and its extensions

The unified AAA design by Symas, and contributed to the OpenLDAP project, replaces the nss-pam-ldapd NSLCD daemon with the SLAPD server, further making available full-fledged SLAPD features such as performance, scalability, and high availability.

The architecture employs OpenLDAP Name-Service Overlay (nssov) which is configured as part of the SLAPD server. The nssov overlay listens to a Unix domain socket for service requests which can be serviced either by database backend(s) or by remote, and possibly foreign, LDAP servers, taking advantage of the connection pooling of the ldap backend as well as the persistent caching capability of the OpenLDAP Proxy Cache engine. Furthermore, a replicated database provides the failover capability during network outage, Figure 3.
Figure 3 The unified AAA architecture using OpenLDAP SLAPD with Name-Service Overlay (nssov), the ldap backend, and proxy cache

The design inherits the architecture enhancements from nss_ldap, pam_ldap, and nss-pam-ldapd, with the added features of compliance and auditing, performance, scalability, and high availability:

- The AAA information can be distributed in a hierarchical and scalable manner, contrasting the flat-file approach.
- The system works alongside with existing solutions, such as NIS, NIS+, DNS, and flat files.
- No application re-compilation or re-linking is needed.
- Standardized APIs are available for authentication and for account, session, and password management.
- The LDAP library is not loaded with every process that accesses the services.
- There is no deadlock in host name lookup of LDAP server during boot process.
- Resource management is available for connections to LDAP servers.
The Proxy Cache extension can be configured to hide access latency and to support disconnected operations.

Replication is possible for high availability and disaster recovery.

Account management can be configured for policy compliance.

Session management can be used for auditing and accounting.

Provisioning AAA Services in Cloud Computing Environments

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [10]. Infrastructure-as-a-Service (IaaS) has become a prevalent service model for cloud computing. Virtual machines are standard deployment units on which infrastructure services can be developed, deployed, and redeployed.

The unified AAA services are implemented and bundled as a software installation package that can be installed on virtual machines and preconfigured as virtual appliances that are deployed as cloud computing instances. Software components, such as secure shell (SSH), invoke industry standard APIs for AAA services. Figure 4 illustrates an example cloud computing deployment which features the following:

- The cloud computing deployment units are virtual appliances preconfigured with an AAA module.
- Each virtual appliance is configured for specific business needs at deployment.
- The security information for each business operation, e.g., online book store and travelling services, is sandboxed within their respective directory store.
- Each virtual alliance allows only the secure shell (SSH) to login to the system, leveraging the security functions provided by the module.
- Virtual appliances can be easily deployed with the provisioned AAA services, when new business units come on board.

- Additional virtual appliances can be dynamically commissioned and decommissioned for a business operation to meet peak workloads.

- The proxy cache configuration brings in relevant business data local to a virtual appliance for high performance.

- The proxy cache also supports disconnected operations for high availability.

- Multiple virtual appliances can further be deployed for high availability.
Figure 4: Provisioning AAA services in an example cloud environment
Provisioning AAA Services in Enterprise Environments

The unified architecture can also be deployed to provision AAA services that meet dynamic enterprise business needs. A sample architecture for enterprise deployments is illustrated in Figure 5. The architecture employs two SLAPD servers as the directory backbone, each serving its respective regional information demands. Both SLAPD servers are configured with the same suffix and with `syncrepl` for replication. Each regional directory server supports multiple clients (organizational units); each client system is configured with a unified AAA module as depicted in Figure 3. The architecture supports the following goals:

- Regional clients are serviced by their respective regional SLAPD servers. Cross-region accesses by clients are reduced to a minimum, if not completely eliminated.
- Cross-region replication can involve less-critical business information for local regions, and therefore can be scheduled for periodical refreshes, instead of continuous synchronization.
- Each client accesses its own sub-tree in the Directory Information Tree (DIT). The modular approach offers scalability for enterprise business growth, as naming space can be provisioned as new clients come on board.
- Each client is equipped with cache configuration. Repeated client accesses will be resolved locally, which eliminates latencies for remote server accesses.
Figure 5 A sample unified AAA architecture, using the ldap backend and proxy cache, for enterprise deployments
High Availability using LDAP Sync Replication Technology

For high availability, additional virtual appliances can be deployed for the same cloud computing instances. Similarly, multiple SLAPD server(s) can be configured for the same business service context in enterprise deployments. Figure 6 shows that two SLAPD servers are configured for the same North America (NA) region. The syncrepl technology is used to keep two regional databases synchronized with each other.

Figure 6 High-Availability Configuration using LDAP Replication Technology

The unified architecture offers a rich pool of failover configuration options. For example, the SLAPD ldap backend configuration provides a URI list that allows specifying a list of servers, e.g., one as the master SLAPD server, while the other failover SLAPD server:
In normal operation mode, the master SLAPD server is used as the information source. LDAP add, delete, and modify requests are sent to the master SLAPD server and replicated to the failover SLAPD server. In case that the master SLAPD server becomes non-operational, the failover SLAPD server can provide the needed business continuity for enterprises. Additional SLAPD server(s) can also be configured for disaster recovery. The disaster-recovery server(s) participates in the replication process, but does not support daily business operations. In other words, the server(s) serves the archival/backup purpose. In disastrous scenarios, the database(s) can be reloaded to the master and failover servers, thereby resuming regular business operations.

3. The nssov Overlay

The OpenLDAP overlays are software components that can be stacked together to customize SLAPD behavior. The nssov overlay offers the additional NSLCD client communication protocol to the SLAPD server. The nssov overlay handles name service and PAM requests through a local Unix Domain socket. The design uses the same nss_ldap and pam_ldap client libraries as those in the nss-pam-ldapd design. The client libraries translate name services and PAM requests into protocol-specific calls to the SLAPD nssov overlay, instead of the nss-pam-ldapd NSLCD daemon. The nssov overlay adopts the schema defined in RFC2307 [11] and RFC2307bis [12].

The nssov overlay may be configured with Service Search Descriptors (SSDs) for each NSS service:

```
nsov-ssd <service> <url>
```

where <service> may be one of the following: aliases, ethers, group, hosts, netgroup, networks, passwd, protocols, rpc, services, shadow. The <url> must be of the form:
The default <basedn> and <scope> are respectively first suffix of the current database and "subtree". The default <filter> depends on which service is being used.

For example, the nssov overlay may be configured with the OpenLDAP ldap backend by adding the following to slapd.conf:

```plaintext
include <path to>nis.schema
moduleload <path to>nssov.la
database ldap
overlay nssov

nssov-ssd passwd ldap:///ou=users,ou=suum,dc=example,dc=com??one
nssov-ssd shadow ldap:///ou=users,ou=suum,dc=example,dc=com??one
nssov-ssd group ldap:///ou=group,ou=suum,dc=example,dc=com??one
nssov-ssd hosts ldap:///ou=hosts,ou=suum,dc=example,dc=com??one
nssov-ssd services ldap:///ou=services,ou=suum,dc=example,dc=com??one
nssov-ssd networks ldap:///ou=networks,ou=suum,dc=example,dc=com??one
nssov-ssd protocols ldap:///ou=protocols,ou=suum,dc=example,dc=com??one
nssov-ssd rpc ldap:///ou=rpc,ou=suum,dc=example,dc=com??one
nssov-ssd ethers ldap:///ou=hosts,ou=suum,dc=example,dc=com??one
nssov-ssd netgroup ldap:///ou=netgroup,ou=suum,dc=example,dc=com??one
nssov-ssd aliases ldap:///ou=aliases,ou=suum,dc=example,dc=com??one
```

If the local database is actually a proxy to a foreign LDAP server, mapping of schema may be needed. Simple attribute substitutions may be performed using the following:

```plaintext
nssov-map <service> <original attribute> <new attribute>
```

The overlay also supports dynamic configuration under "cn=config". The layout of an example configuration entry is as follows:

```plaintext
dn: olcOverlay={0}nssov,ocDatabase={1}hdb,cn=config
objectClass: olcOverlayConfig
objectClass: olcNssOvConfig
olcOverlay: {0}nssov
olcNssSsd: passwd ldap:///ou=users,ou=suum,dc=example,dc=com??one
olcNssMap: passwd uid accountName
```

which enables the passwd service and uses the accountName attribute to fetch what is usually retrieved from the uid attribute.
PAM authentication, account management, session management, and password management are supported. Authentication is performed using LDAP simple binds.

Two methods of mapping a PAM user name to an LDAP Distinguished Name (DN) are provided:

- The SLAPD's authz-regexp facility - In this case, a DN of the form
  
  `cn=<service>+uid=<user>,cn=<hostname>,cn=pam,cn=auth`

  is fed into the regexp matcher. If a match is produced, the resulting DN is used.

- The NSS passwd map

If no DN is found, the overlay returns PAM_USER_UNKNOWN. If the DN is found, and password policy is supported, a simple bind will use the password policy control and return expiration information to PAM.

Account management also uses two methods. These methods depend on the ldapns.schema included with the nssov source:

- The first method is identical to the one used in PADL's pam_ldap module. The host and authorizedService attributes of a user’s entry is checked to determine access permissions. Access can also be controlled based on a user’s group membership. This method is much less flexible and doesn't scale well to large networks of users, hosts, and services.

- The second method uses SLAPD's Access Control (ACL) engine to check if a user has compare privilege on an ipHost object whose name matches the current host name, and whose authorizedService attribute matches the current service name. This method is preferred, since it allows authorization to be centralized in the ipHost entries instead of scattered across the entire user population.
The ipHost entries must have an authorizedService attribute defined by the authorizedServiceObject auxiliary class.

For session management, the overlay may optionally add a "logged in" attribute to a user’s entry for successful logins, and delete the corresponding value upon logout. The attribute value is of the form

<generalizedTime> <host> <service> <tty> (<ruser@rhost>)

The overlay performs password management by issuing a PasswordModify LDAP extended operation in the server for a given user.

4. The Proxy Cache Engine

The Proxy Cache engine of SLAPD [3] was designed to improve the responsiveness of the ldap and meta backends. Instead of caching only individual entries, the proxy cache stores data and semantic information corresponding to recently answered queries. The cache manager implements the following three algorithms:

- Query containment algorithm decides whether an incoming search request is semantically contained in any of the recently answered queries, e.g., (shoesize >= 9) is contained in (shoesize >=8). A contained query is answerable from the cache.
- Cache replacement algorithm determines when a query and entries should be removed from the cache.
- Consistency control algorithm maintains the consistency between cached data and the corresponding information stored in persistent data store.

The proxy cache handles a search query by first determining whether it is contained in any cached search filter. Contained requests are answered from the proxy cache’s local database. Other requests are passed on to the underlying ldap or meta backend and processed as usual. The LDAP matching rules and syntaxes are used while comparing assertions for query containment.
To simplify the query containment implementation, a list of cacheable templates (defined below) is specified at configuration time. A query is cached or answered only if it belongs to one of these templates. The entries corresponding to cached queries are stored in the proxy cache local database while their associated meta information (filter, scope, base, attributes) is stored in main memory.

A template is a prototype filter for generating LDAP search requests. Search filters are templates associated with their respective list of attribute values. Prototype filters are similar to those defined in [13], except that the assertion values are missing. Examples of prototype filters are (sn=) and (&(sn=)(givenname=)) which are instantiated by search filters (sn=Doe) and (&(sn=Doe)(givenname=John)) respectively.

The cache replacement policy removes the least recently used (LRU) query and entries belonging to only that query. Queries are allowed a maximum time to live (TTL) in the cache thus providing weak consistency. A background task periodically checks the cache for expired queries and removes them.

The following directive enables proxy caching and defines the configuration parameters:

```
proxycache <db> <maxentries> <nattrsets> <entrylimit> <period>
```

The <db> parameter specifies the underlying database type which is used to hold the cache entries. The <maxentries> parameter specifies the total number of entries that may be held in the cache. The <nattrsets> parameter specifies the total number of attribute sets that may be defined. The <entrylimit> parameter specifies the maximum number of entries in a cacheable query. The <period> parameter specifies the consistency checking period (in seconds). In each check period, queries with expired TTL are removed.

The proxyAttrSet directive is used to associate a set of attributes to an index:

```
proxyAttrSet <index> <attrs ...>
```
The proxyTemplate directive further associates a cacheable prototype filter and the time-to-live (TTL) parameter with an index of an attribute set:

```
proxyTemplate <prototype filter> <attrset_index> <TTL>
```

The following sample SLAPD configuration defines the proxy attribute sets and proxy template for user passwd and group services.

```
#overlay proxycache
proxycache bdb 100000 11 1000 100
# posixAccount
proxyAttrset 0 cn uid uidNumber gidNumber homeDirectory userPassword loginShell gecos
description objectClass
# shadowAccount
proxyAttrset 1 uid userPassword shadowLastChange shadowMin shadowMax shadowWarning
shadowInactive shadowExpire shadowFlag description objectClass
# posixGroup
proxyAttrset 2 cn gidNumber userPassword memberUid uniqueMember description objectClass

# proxy templates
proxyTemplate (&(objectClass=)(uid=)) 0 3600
proxyTemplate (&(objectClass=)(uidNumber=)) 0 3600
proxyTemplate (objectClass=) 0 3600
proxyTemplate (&(objectClass=)(uid=)) 1 3600
proxyTemplate (&(objectClass=)(cn=)) 2 3600
proxyTemplate (objectClass=) 2 3600
proxyTemplate (&(objectClass=)(gidNumber=)) 2 3600
proxyTemplate &(objectClass=)((memberUid=)(uniqueMember=)) 2 3600
```

5. LDAP Sync Replication

Replicated directories are a fundamental component of a robust IT infrastructure. The OpenLDAP replication involves providers and consumers. A provider replicates directory updates to consumers which in turn can also serve as providers and propagate updates to other consumers.

The LDAP Sync Replication (syncrepl) [2] is a consumer-side replication engine which executes as one of the SLAPD threads. The replication engine maintains a consumer replica by connecting to a replication provider to perform either periodic content pooling.
or persistent request for timely updates upon content changes. The syncrepl provider is implemented as an overlay.

Syncrepl uses the LDAP Content Synchronization protocol [14] which defines a set of controls and protocol elements that extend the LDAP search operation. Two synchronization modes are supported:

- `refreshOnly` – The consumer polls the provider using an LDAP search request with an LDAP Sync control. The provider returns all the entries matching the search criteria. The consumer copy is synchronized with the copy of the provider at the time of polling. The synchronization operation is completed until the next scheduled polling request.

- `refreshAndPersist` - The synchronization begins with an LDAP search request from the consumer to the provider, like `refreshOnly`. After the provider returns all entries matching the search criteria to the consumer, the synchronization search remains persistent in the provider, while the consumer listens for further updates from the provider. Subsequent updates to the synchronization content in the provider cause additional entries to be sent to the consumer.

The OpenLDAP implementation of the LDAP Sync Replication specifications offers the flexibility for various deployment alternatives:

- Delta-Syncrepl – LDAP Sync Replication is an entry-based replication. When an attribute value of an entry in the provider is changed, the entire entry is fetched by the consumer. The worst-case scenario of this replication scheme is when there are a huge number of entries, each of which with very minor attribute value changes. With delta-syncrepl, a change log is maintained in the provider. The replication consumer fetches changes from the change log in the provider and applies the changes to its own database. If a consumer is too much out of sync, the conventional syncrepl scheme comes into play.

- N-Way Multi-Master – The replication design replicates data to multiple providers, potentially on multiple physical sites. The design avoids single-
point of failure and offers the possibility for automatic failover and high availability.

- **Mirror Mode** – This is a hybrid configuration that aims at providing data consistency in a single-master deployment and the high availability of a multi-master deployment. The design employs a frontend that distributes updates among multiple providers. The frontend makes sure that each update is only sent to one provider which replicates to other providers.

- **Syncrepl Proxy Mode** – Both refreshOnly and refreshAndPersist modes of synchronization must be initiated from the consumer. In some network configuration, e.g., where firewalls restrict the direction of network connection, provider-initiated replication is needed. Syncrepl proxy mode is for such purpose.

6. **Deployment Alternatives using LDAP Sync Replication vs. Proxy Cache**

The unified AAA design provides the flexibility in that the client systems can be configured using a database backend with syncrepl, replacing the ldap backend with proxy cache, Figure 7. The client systems are configured as the syncrepl consumers which synchronize with the providers in the directory backbone.
Figure 7: A sample unified AAA architecture using a database backend and syncrepl
The following sample slapd.conf sets up a syncrepl provider overlay in a directory backbone:

```
# bdb backend as a syncrepl provider
database   bdb
suffix     dc=example,dc=com
rootdn     dc=example,dc=com
directory  /var/ldap/db
index      objectclass, entryCSN, entryUUID eq

overlay    syncrepl
syncrepl-checkpoint  100  10
syncrepl-sessionlog  100
```

The following slapd.conf file configures a client system as a syncrepl consumer, using the refreshOnly replication mode:

```
database     hdb
suffix       dc=example,dc=com
rootdn       dc=example,dc=com
directory    /var/ldap/db
index        objectclass, entryCSN, entryUUID eq

syncrepl     rid=<replica ID>
             provider=ldaps://provider.example.com:636
             type=refreshOnly
             interval=01:00:00:00
             searchbase="dc=example,dc=com"
             filter="(objectClass=organizationalPerson)"
             scope=sub
             attrs="cn,sn,l,ou,uid,uidNumber,gidNumber"
             schemachecking=off
             bindmethod=simple
             binddn="cn=syncmgr,dc=example,dc=com"
             credentials=secret
```

The client system is configured as a syncrepl consumer that will connect to the provider SLAPD at port 636 on the system, provider.example.com, using LDAP over SSL/TLS. The connection uses simple authentication method, via encrypted channel, with the bind DN “cn=syncmgr,dc=example,dc=com” and password “secret”.

Note that the privilege of the user DN “cn=syncmgr,dc=example,dc=com” should be set properly in the provider to retrieve the replication content. In addition, the search limits need to be set high enough for the consumer to retrieve the entire replication content. The synchronization search in the running example request entries whose objectClass
is organizationalPerson in the entire subtree under “dc=example,dc=com”. The requested attributes are “cn,sn,l,ou,uidNumber,gidNumber”. The schema checking is turned off so that the client SLAPD will not enforce entry schema checking when it processes updates from the provider SLAPD.

7. Summary

This paper presented a unified architecture for provisioning AAA services in cloud and enterprise computing environments. The design offers performance, scalability, and high availability. In addition, the flexible design allows AAA services to be provisioned modularly and dynamically based on business demands. The architecture has been based on open standards and evolved with the collective efforts of the open source community for the past many years. The implementation is contributed back to the open source community and freely available.

References


http://www.openldap.org/, December 20, 2009

http://www.openldap.org/pub/kapurva/proxcaching.pdf


Biographies

Ted C.-M. Cheng serves as a consultant at Symas Corporation. His R&D interests are in on-demand computing, provisioning internet services in cloud and enterprise environments, performance, scalability, and high availability. He received his Ph.D. degree in EE-Systems from the University of Southern California in 1993 and has been in enterprise computing and open source development ever since. He is a member of IEEE.

Howard Chu is the Chief Architect of OpenLDAP and CTO of Symas Corporation. Prior to founding Symas Corporation, Howard worked at the U. Michigan, JPL, Locus Computing, and platinum Technology in software development roles. Howard is a prolific contributor to the Open Source software community.

Matthew Hardin is a co-founder of Symas Corporation and currently serves as its VP of Product Engineering. Throughout his 30-plus-year career he has acquired in-depth experience with embedded OS development, communications, server, and client software development in a wide variety of environments.