SUN BLUEPRINTS
[illegible]
APPROVED

Implementing LDAP in the Solaris™ Operating Environment

By Tom Bialaski - Enterprise Engineering

Sun BluePrints™ OnLine - October 2000

Sun microsystems
ENTERPRISE
ENGINEERING

http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-7036-10
Revision 01, October 2000

Implementing LDAP in the Solaris™ Operating Environment

Abstract

The first new naming service in over 10 years was introduced in the Solaris™ 8 Operating Environment. This naming service is based on the Lightweight Directory Access Protocol (LDAP) which provides an industry standard interface for accessing data stored in LDAP compliant directories.

This paper takes a look at how LDAP was implemented in the Solaris 8 Operating Environment and what steps you need to perform to take advantage of this technology. Since the installation and configuration of native Solaris LDAP is quite complex, only an overview of what steps are required is presented.

Introduction

The acronym LDAP stands for Lightweight Directory Access Protocol, but LDAP is more that just a protocol. Besides providing a standard access method for clients and servers, LDAP also defines a naming, information, and security model for how data is stored and protected. Knowing how these models work and how they pertain to the Solaris LDAP implementation will help you gain a better understanding why certain configuration steps are required.

The first part of this whitepaper explains what the LDAP models are in order to form a foundation for the second part which delves into the actual implementation. While not meant to be an all inclusive guide to deploying LDAP as a naming service, this whitepaper should serve as a foundation for understanding the fundamental principles involved.

Although the Solaris native LDAP implementation is independent of any particular LDAP server, the server chosen for these examples is the iPlanet™ Directory Server 4.11 from iPlanet E-Commerce Solutions, a Sun-Netscape Alliance. The server software ships on a companion CD co-packaged with the Solaris 8 Operating Environment.

LDAP Models

LDAP is commonly defined in terms of four models. These are:

- Information Model
- Naming Model
- Access Model
- Security Model

The information model specifies how data is stored within the directory. The naming model defines how objects within the directory structure are organized and identified. The mechanics for accessing data are specified in the access model. How data is protected is defined in the security model.

The following sections examine what each of these models consists of and how they are implemented in native Solaris LDAP.

Information Model

The basic LDAP storage unit is the directory entry, which is where information about a particular object resides. Objects are a collection of attributes which each have a corresponding value. What attributes an object may contain is defined in an object class. For example, to describe a person, an object of object class `person` is created. The person object class defines a set of attributes, like first name, surname, and telephone number, which describes the person you are creating a directory entry for.

To maintain order, a set of rules is established to govern which attributes are required, which ones are optional, and what type of data can be stored in them. This set of rules is called the directory *schema*. To promote interoperability between different vendor's LDAP servers a well-defined *standard* schema exists, which is expected to be included on all LDAP servers.

Within the LDAP standard is a provision for extending the schema. That is, to create your own object classes and attributes. When you develop new LDAP-enabled applications, you are advised to use the standard object classes if possible. Sometimes, as was the case with native LDAP, new object classes and attributes are required.

To extend the schema which ships with the iPlanet Directory Server 4.11, you update the schema configuration files and then restart the server. This step is required before entries which support native Solaris LDAP can be created.

Implementing LDAP as a Solaris naming service requires making the same information that NIS maps contain available to naming service clients. LDAP entries that represent the data found in NIS maps must be created. To promote interoperability, the object class definitions used to store NIS data are defined in a specification called RFC 2307 which was later updated to RFC 2307bis.

The RFC 2307 schema definitions are part of the iPlanet Directory Server 4.11 default configuration, however, there are some object classes that go beyond the RFC2307 specification which do need to be defined.

In addition to LDAP entries that contain NIS map data, object classes which are required to support the Solaris LDAP client implementation must be defined in the schema. Information about how the client connects to a server is placed in a client profile. Also, each LDAP server which supports Solaris clients must have an entry which denotes the name of the domain that it is servicing.

The specific object classes required to support native Solaris LDAP and their associated attributes are described later in this whitepaper.

Naming Model

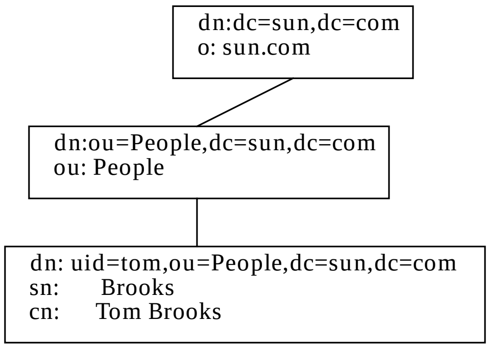
How entries are organized within a directory is defined by the naming model. The tree-like structure that information is kept in is called the directory information tree (DIT) which looks similar to a Solaris filesystem. However, there are a number of subtle but important differences.

Unlike a Solaris filesystem there is no root directory, which serves as an entry point into the entire structure. Instead, LDAP directories contain one or more suffixes which signify the top node of a DIT. Under each suffix there is a separate DIT which provides its own namespace. Each directory server does have an entry called the Directory Specific Entry (DSE) which contains information pertinent to the directory server but is not connected to any of the DITs.

Below the directory suffixes are containers and directory entries. These are similar to filesystem directories and files except that containers can hold data in addition to entries and other containers. In a filesystem, only files contain data.

Entries are identified by their distinguished name (DN) which is similar to a filesystem absolute path name. The main difference is that the DN is specified in the reverse order of a filesystem pathname.

The following diagram shows a portion of a DIT to illustrate how DNs are referenced.


dn:dc=sun,dc=com
o: sun.com
dn:ou=People,dc=sun,dc=com
ou: People
dn: uid=tom,ou=People,dc=sun,dc=com
sn: Brooks
cn: Tom Brooks

In this example, the suffix of the DIT is `dc=sun,dc=com`. Below the suffix is a container called `ou=People` and below the container is a directory entry for a person in the organization.

As you can see, the DN of the person is specified beginning with an attribute value, followed by a container, then the suffix. An interesting note here is that the entry is identified by an attribute, which is `uid` in this case. Any attribute can be specified, but since some entries may have common attribute values, for example, `manager=`, one attribute is chosen whose value will be unique to each entry. For user account entries, the `uid` attribute is used.

Alternatively, a relative distinguished name (RDN), which is similar to a filename can be used to identify an entry. In this example, `uid=tom` would be the RDN which refers to the entry. Since there is only one entry which has a value of `uid=tom`, the DN does not need to be specified to identify the entry.

Another aspect of naming is the abbreviations used to identify suffixes and containers. Suffixes can appear either in the X.500 style or the domain component style. X.500 specifies a rigid nomenclature which includes a country code, locality, and organization name. The domain component (`dc`) style mirrors the DNS address of a company and is preferable to the X.500 style since registered DNS names are guaranteed to be unique. Instead of using the dot notation, a series of comma-separated `dc=` statements are specified.

To configure an LDAP server to support native Solaris LDAP clients, you need to create a DIT to provide a structure to store the NIS map data. You can store data under any branch in the DIT but the client must be aware of where to start looking for it. The top level of the DIT must also have a `nisDomainObject` entry which holds the identity of the domain the server is servicing.

Access Model

The access model defines how a client gains access to the directory and subsequently starts issuing requests. Before any search or write requests can be made, the client must be authenticated by the directory server. After authentication a connection is established which is used for subsequent LDAP requests.

The authentication process is called *binding* and consists of the client sending the directory server a set of credentials and the server sending back either a success or failure error code. The security method used can either be a simple username password pair or a more sophisticated method such as a challenge/response mechanism.

Once the client is authenticated, commands for searching and modifying entries can be sent. Search commands are the most common, consisting of specifying a search base (where in the DIT to start searching) and a particular attribute value or wildcard. The server either returns a list of matching entries or a `NOTFOUND` error.

To disconnect from the server, the client sends an `unbind` command, which closes the connection.

When LDAP is used as a Solaris naming service, the Solaris system takes on the role of a LDAP client. During the booting process, the Solaris client binds to the directory server using a specified security method to establish a connection. Once the connection is established, the Solaris client sends LDAP commands and retrieves data back from the directory. System utilities that use the Solaris name switch, issue `getXbyY` type calls which in turn get translated to LDAP commands.

The information the Solaris client uses to bind to the server, resides in a profile file which gets read initially when the system boots and then is placed in a cache. The client periodically checks to see if the profile it is using gets updated on the server. If it does, the cache is refreshed with the new information and, if need be, the client will automatically bind to another directory server.

The iPlanet Directory Server 4.11 can only support simple username/password authentication from Solaris LDAP clients, although the client can support more sophisticated methods if they were available.

Security Model

The security model defines how objects in the DIT are protected and who is authorized to access them. This model is very flexible allowing for the protection of the entire DIT down to the attribute level. You can set Access rights for a single user, a group of users, all users, or anonymous users. The model distinguishes between all users, that is, users with an account on the server, and anonymous users, who do not.

The user identity is determined by the DN that the client binds to the directory with. The DN usually contains the `uid` of the person binding to the directory. If that person appears in a group, then group access rights apply.

Access rights are established by specifying an access control instruction (ACI) which appears as an attribute in a directory entry. ACIs can be set to either allow or deny access. Multiple rules can be defined within a single ACI.

To use LDAP as a Solaris naming service, specific access rights need to be established. Most of the data stored needs to be readable by everyone and only writable by designated administrators. However, the `userpassword` attribute needs to be modifiable by the owner.

Solaris LDAP Naming

This section describes the specific naming contexts used in the native Solaris LDAP implementation starting at the DIT top node and working down to NIS object names.

The suffix of the DIT that supports Solaris LDAP is best represented by using the domain component (`dc`) nomenclature. The `dc` is typically a sub-component of your DNS name. For example, a directory server at Sun might have the suffix: `dc=east,dc=sun,dc=com`. Alternatively the suffix may be expressed as an organization (`o=`), but the `dc=` notation is more convenient since most DNS names are aligned with NIS domain names.

Located somewhere below the top node in the DIT are several containers which are referenced using the Organizational Unit (`ou=`) notation. These are:

- ou=people

Stores login and password information similar to `/etc/password` and `/etc/shadow.` The objects stored here are `posixAccount` and `shadowAccount`.

- ou=group

Stores Solaris group information, similar to `/etc/group`. Objects of the type `posixGroup` are stored here.

- ou=services

Stores information about available services, similar to `/etc/services`. Objects of the type `ipService` are stored here.

- ou=protocols

Stores information about protocols, similar to `/etc/protocols`. Objects of the type `ipProtocols` are stored here.

- ou=rpc

Stores information related to remote procedure calls (RPCs) similar to `/etc/rpc`. Objects of the type `oncRPC` are stored here.

- ou=hosts

Stores the host table, similar to `/etc/hosts`. Objects of the type `ipHost` are stored here.

- ou=ethers

Stores ethernet addresses, similar to `/etc/ethers`. Objects of the type `ieee802Device` and `bootableDevice` are stored here.

- ou=networks

Stores names of networks, similar to `/etc/networks`. Objects of the type `ipNetwork` are stored here.

- ou=netgroup

Stores netgroup information in the object type `nisNetwork`.

- ou=profiles

Stores LDAP client profiles in the object type `SolarisNamingProfile`.

- ou=projects

Stores project accounting information in the object type `SolarisProject`.

- ou=solarisauthattr

Stores information used in Role-based Access Control authentication.

- ou=solarisprofattr

Stores information used in Role-based Access Control authentication.

- nismapname=auto_*

Stores automounter information.

Solaris LDAP Schema

The schema required by Solaris LDAP clients consists of object classes that come with iPlanet Directory Server 4.11 and some additional ones. These object classes and their attributes are listed below.

Object classes:

- posixAccount

Requires `cn`, `uid`, `uidNumber`, `gidNumber`, and `homeDirectory`. Optional attributes are `description`, `gecos`, `loginShell`, `userPassword`

- shadowAccount

Requires `uid`. Optional attributes are `description`, `shadowLastChange`, `shadowMax`, `shadowMin`, `shadowWarning`, `shadowInactive`, `shadowExpire`, `ShadowFlag`, and `userPassword`.

- posixGroup

Requires `cn` and `gidNumber`. Optional attributes are `description`, `memberUid`, and `userPassword`.

- ipService

Requires `cn`, `ipServiceProtocol`, and `ipServicePort`.

- ipProtocol

Requires `cn` and `ipProtocolNumber`.

- oncRPC

Requires `cn` and `oncRpcNumber`.

- ipHost

Requires `cn` and `ipHostNumber`. Optional attributes are `bootFile`, `bootParameter`, `description`, `macAddress`, `manager`, `serialNumber`.

- ipNetwork

Requires `cn` and `ipNetworkNumber`. Optional attributes are `description`, `ipNetmaskNumber`, and `manager`.

- nisNetgroup

Requires `cn`. Optional attributes are `description`, `memberNisNetgroup`, and `nisNetgroupTriple`.

- ieee802Device

Requires `cn` and `macAddress`.

- bootableDevice

Requires `cn`. Optional attributes are `bootFile` and `bootParameter`.

- nisMap

Requires `nisMapName`. Optional attribute is `description`.

- nisObject

Requires `nisMapName`. Optional attributes are `cn`, `description`, and `nisMapEntry`.

- nisKeyObject

Requires `cn`, `nisPublicKey`, and `nisSecretKey`. Optional attributes are `uidNumber` and `description`.

- nisDomainObject

Requires `nisDomain`.

Solaris LDAP Software

The Solaris 8 Operating Environment implementation is client side only. That is, the client expects to see a properly configured LDAP server. As described in the next section, you need to perform particular modifications to the LDAP server configuration to support Solaris LDAP clients.

The specific pieces of software that comprise the Solaris LDAP implementation are:

- LDAP client
- PAM module
- `nsswitch.conf` `ldap` tag
- LDAP libraries
- LDAP tools

The LDAP client is the piece of software which is run from the Solaris startup scripts in place of, or in addition to, `ypbind`. The client software is responsible for reading a configuration file that provides instructions on *what* LDAP server to connect to and *what* credentials to use for authentication. Once the client is running, the `ldap_cachemgr` daemon is responsible for updating the client profile with a current copy. This is accomplished by periodically checking the profile configuration file on the LDAP server to see if it has changed.

The UNIX® PAM module has been modified in the Solaris 8 Operating Environment to work with data stored in an LDAP directory. When this module is used for user authentication, passwords are stored in crypt format on the directory server like they would in the NIS or NIS+ data stores. The authentication is then performed locally on the client system after the crypted password is retrieved. A new PAM LDAP module is also available in the Solaris 8 Operating Environment. This module uses authentication methods that may be available on the LDAP server, such as CRAM-MD5. Instead of being performed locally, authentication takes place on the LDAP server.

The Name Service Switch has been enhanced to include the `ldap` tag as an option. LDAP can be used as the only naming service or as a supplemental one. The same rules for naming service searches apply.

LDAP libraries are included so LDAP-enabled applications, such as Solaris LDAP tools can use them. The libraries can also be used to create your own LDAP-enabled applications.

The standard `ldapmodify` and `ldapsearch` commands are available in the native Solaris LDAP package. To view data stored in a LDAP directory, the command `ldaplist` is provided which performs a similar function as the `ypcat` command.

Server Configuration Changes

Any LDAP server that supports specific V3 features, such as Virtual Lists View (VLV) and page mode, can be configured to support Solaris LDAP clients, but since the iPlanet Directory Server is bundled with Solaris 8 Operating Environment, it is used as the sample LDAP server. To configure your LDAP server perform the following steps.

- Add additional schema elements
- Set proper permissions
- Set up the DIT
- Populate the DIT
- Create a `proxyagent` account
- Tune for performance

Adding Schema Elements

As described in the schema section, there are a number of LDAP object classes and associated attributes which must be defined in the LDAP server configuration files. In the iPlanet Directory Server 4.11, some of the object classes and attributes ship with the standard schema files. These are the RFC 2307 defined object classes which specify NIS to LDAP mappings. You need to add additional object classes which are part of an updated version of RFC 2307 and also add object classes which support Solaris LDAP client profiles.

The easiest way to update the schema files is to obtain the new object class and attribute files and copy them into the user-defined schema files.

Setting Proper Permissions

Correct permissions need to be established for the objects residing in the DIT. This requires modifying the default ACIs for particular directory objects. You need to modify the ACI, VLV Control. The Virtual List View (VLV) feature is used by the Solaris LDAP client to improve performance. The Solaris client requires anonymous access to the VLV object, which is not the default, so it needs to be changed.

As a Solaris naming service, you want information to be readable by everyone, but only modifiable by certain administrators. One exception to this is a user’s password, which needs to be modifiable by the owner of the object.

The ACI modification can either be made through the iPlanet Directory Console or by importing a LDAP file which contains the proper ACIs.

Setting up the DIT

Before data can be placed in the LDAP directory, a proper DIT structure must exist. Create the containers, from the iPlanet Directory Console, for the NIS map data (which will be imported) or by importing a LDIF file.

Populating the DIT

Included on the iPlanet companion CD which ships with the Solaris 8 Operating Environment is an utility called `dsimport`. This utility takes input in the form of `/etc` files then creates the appropriate LDAP entries. A `nis.mapping` file is used to customize where and how the data is placed in the DIT.

Creating a ***proxyagent*** Account

Before a Solaris LDAP client can bind to a LDAP server, it must be authenticated. This requires the existence of a LDAP account for the client. In reality, a LDAP account is any entry that contains an `userPassword` attribute.

By default, an entry called `proxyagent` is used by the Solaris LDAP client as the account it binds as. This account must be set up and granted appropriate permissions so a Solaris LDAP client can access LDAP as a naming service.

Tuning for Performance

Once the basic configuration steps are performed on the LDAP server it is recommended that the server be tuned to provide better performance, especially when dealing with large maps. To achieve optimum performance, database indices are created for commonly accessed attributes.

LDAP Client Setup

The client setup consists of running a client program that binds to a specified LDAP server, then accesses configuration data found on the server. The result of running the initialization program is the creation of two configuration files. One file contains the client's credentials and the other file contains information about the server it is connecting to and other configuration parameters. These files are located in `/var/ldap` and are called `ldap_client_cred` and `ldap_client_file`.

The other configuration parameter which needs to change is in the `nsswitch.conf` file. Depending on which naming service you want to use, and in which order you choose to search them in, add the `ldap` tag to the appropriate lines. The client initialization program automatically makes these changes.

The other option which needs to be set is the type of authentication you want to use. The two options are: `pam_unix` and `pam_ldap`. The `pam_unix` module performs the authentication locally, while the `pam_ldap` module preforms it on the LDAP server.

Alternatives

You can deploy LDAP without running the Solaris LDAP client. The NIS extensions for Solaris Operating Environment is an alternative which provides a NIS server front-end to a LDAP directory and a synchronization between NIS maps and LDAP entries. The implementation is transparent to NIS clients, which do not have to be modified. The drawback with this approach is that you need to maintain both NIS maps and the LDAP directory.

An alternative to the NIS extensions is the `ypldapd` software from PADL. This software provides a gateway between NIS and LDAP. In this implementation, there is only one data store which is the LDAP directory. The drawback here is that Sun service does not provide support for `ypldapd`.

Conclusion

Implementing LDAP as a Solaris naming service is not an easy task. Since LDAP provides a general purpose directory, it is very flexible. However, with flexibility comes complexity. While it does not make sense transitioning from NIS/NIS+ just for the sake of doing it, the future benefits of a consolidated data store, makes it worth exploring.

References

Solaris and LDAP Naming Services - Sun BluePrints

Understanding and Deploying LDAP Directory Services - Howes, Smith, Good

Acknowledgments

I would like to thank Michael Haines, co-author of the *Solaris and LDAP Naming Services* Sun BluePrints for all his help in locating correct information.

Author’s Bio: Tom Bialaski

Tom Bialaski is currently a Staff Engineer with the Enterprise Engineering group at Sun Microsystems, and is the author of "Solaris Guide for Windows NT Administrators." Tom has nearly 20 years of experience with the UNIX operating system and has been a Sun Engineer since 1984.