OpenLDAP as a Web Application Database

TT1138

Johnnie Odom
Systems Integration Manager
School District of Escambia County
jodom@escambia.k12.fl.us

A Brief Introduction

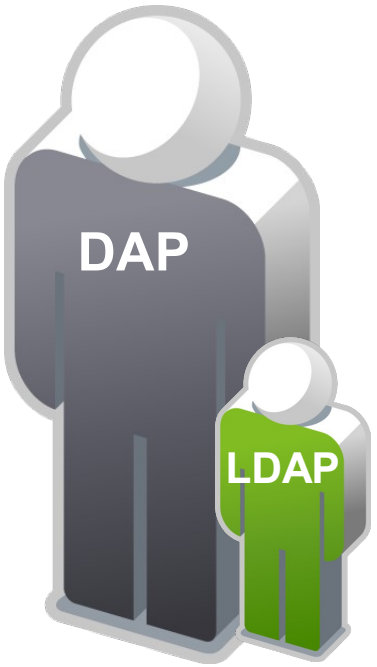
- A Few Theatrical Guidelines
  - One person's “Professionals Only Do This” is another's “Professionals Never Do This”
  - The range of techniques in your toolkit is as important as your skill in implementing them.
  - This lesson is from the Systems world
    - You are better programmers than me.

The LDAP Ecosystem

Mom, What's LDAP?

- ***Lightweight*** Directory Access Protocol (v3)
- Most common implementations are Microsoft Active Directory, Novell eDirectory, OSS OpenLDAP, and Apple Open Directory
  - Those last two are the same thing.
  - Somewhere you might find implementations from Netscape, SUN, and IBM floating around.
- Most syadmins think of it as a way of keeping track of users, groups, and computers.
- Most programmers think of it as something to authenticate against.
- Most users know it as an address book.

Pygmy Children of a Giant Race



- LDAP is a subset of DAP.
- DAP is part of the X.500 specification.
- So are X.509 certificates
- X.500 is very difficult to implement in full.
- Even though LDAP is lightweight, it borrows items from the full X.500 as needed.
- As the use of LDAP strives towards full X.500, richer (but more complicated) functionality is exposed.

A Theoretical Application

- Network Address Tracker
  - Ipv4 represented by 4 integer fields
  - Informal name, DNS name
  - Comment field
  - Derive location from placement in tree

Authorization controlled by users and groups internal to the application.

  - Create administrator contact lists dynamically from user info.

- Written in PHP, but we could use almost anything.

What You Will Need

- An instance of OpenLDAP
- A programming language or platform
  - With an LDAP library.
- A web server supporting the environment above.
- Your favorite code editor.
- An LDAP Tool of some sort.
  - LDAP Browser
  - Apache LDAP Studio
  - PHPLdapAdmin
- SSL may or may not be required.

Your beverage of choice

Data Types (Simple Version)

- Strings
- Numbers
- Dates
- Network Addresses
- XML
- Binary Blobs (including pictures)

Data Types (Complex Version)

Audio

Binary

Bit String

Boolean

Certificate

Certificate List

Certificate Pair

Country String

Directory String

Distinguished Name

Facsimile Telephone Number

Generalized Time

IA5 String

Integer

JPEG

Name And Optional UID

Numeric String

Octet String

OID

Other Mailbox

Postal Address

Printable String

RFC2307

RFC2307 Boot Parameter

Serial Number and Issuer

Supported Algorithm

Telephone Number

Telex Number

Stages of Program Creation

1. Determine objects and hierarchy
2. Create new objects and attributes via LDIF
3. Manually create starter objects
4. Implement Authentication
5. Set up LDAP Connection
6. Implement Read Functions
7. Implement Write Functions
8. Debug and Iterate Functionality

Determine Objects and Hierarchy

- What is already built-in?
- Containers (OU)
  - Separate Objects for Security
  - Mirror Organizational Structure
  - Enforce Hierarchy or Tree Structure
- Users, Groups, Machine Records
  - Follow Standards
  - Handle Housekeeping and Access
- ACLs

Create New Objects, Attributes in LDIF

- Objects can inherit from each other.
- Reuse attributes wherever possible
- Create minimum needed
  - Cost for adding later is minimal
- Single and multi-valued attributes
- Schema extension and object creation both possible via LDIF.

Structuring Data

- Psuedo-Pointers implemented by holding Dns as values.
- Double-linking (members and membership) is effective for lookups, but does require maintenance.
- Implement attribute indexes, but carefully.
- Floating-Point values have to be implemented in other data types (binary, etc.)

Manually Create a Starter Objects

- Prime the pump by creating some ideal objects and configurations.
- After development ends, setup can be via LDIF and script.
- Use LDAP tool of your choice or CLI.

Set Up LDAP Connection (I)

- Can use a single reader / writer or can use a reader and then do a full bind with actual user.
- Connections and objects retrieved from authorization should be stored in session state.

Program Flow might look like this:

- Connect
- Bind (1) for User DN Lookup: Anonymous or Generic User
- Search for User DN
- Bind (2) As User DN: Automatic Rights Handling
  - Can Re-USe
- Search for Entries
- Get Specific Entry
- Read Entry Attributes via Search
- Add or Delete Entries
- Modify or Replace Attributes
- Compare Attributes
- Rebind or Unbind and Destroy Connection

Set Up LDAP Connection (II)

- Connect, not Bind, is the reference passed to LDAP calls to refer to this instance.
- Connect does not necessarily test for server existence. That might require bind depending on library.
- To reuse a connection, most libraries just let you rebind with the same connection.
- Unbind usually both unbinds and destroys connection. Destroy calls usually a synonym for unbind.

Implement Authorization

- Bind with username / password and return true or false for authorization.

DON'T STOP THERE

- Grab ALL user attributes. Inexpensive operation but can yield useful results.
- Scan key objects for configuration and internal authorization (groups and any custom configuration objects).
- Set a login timeout with an unbind call.

Implement Reads

- All reads are search queries followed by picking up one or more objects from search results.
- ***ALWAYS*** filter searches for object type and location and then later by names and other attributes.
  - Often different objects in various contexts have similar names.
  - Can also filter via attributes.
- Queries, even large ones, should be very fast.
  - Beware of implementation-specific query limits.

More On Reads

- Be as specific as possible without missing possible matches.
- **Options:** Scope (BASE, ONE, SUBTREE)
- **Option:** Filter (Complex Topic See Documentation)
- **Option:** Attributes to retrieve
- DN is not searchable
- Reads are performed in conjunction with returning search results.
- Entries (objects) read by DN as part of Search
- Most reads on attributes
- Reading attributes follows a standard key (attribute) / value approach.
- Hierarchy not necessarily represented as attributes
- LDAP compare probably better than reading data and doing compare.

Data Sanitization

- For reads, biggest risk is over-querying. Possible for users to query a wild card.
- If using a user with ACLs, possible to restrict results instead of giving a super-user write abilities.
- LDAP will happily store weird values.

Implement Writes

- Options for Writes:
  - Add
  - Delete
  - Move
  - Modify, with options:
    - Add
    - Delete
    - Replace
- Be precise on write location.
- No batch operations.
- Beware multi-valued attributes.

Iterate and Debug

- Review: Attributes can be added easily without changing current objects.
  - Still need to check for nulls or missing attributes.
- Bugs likely not system-destroying.
  - Never a need to implement non-loop deletes.
- Lots of logic in code layer.
- May want to create separate housekeeping processes. Background reads for checking will have minimal impact.

What Kinds of Utility Functions?

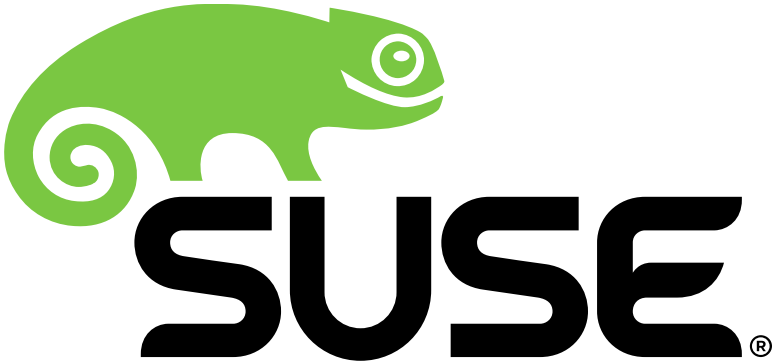
- Time Conversion
- Casting (String to Num)
- Error Code Translation
- Objects for More Complex Operations or Multiple Operations
- DN to CN Convversions
- Context Chopping
- Atomic Operations (Double-Linking)

Server-Specific Details

- Back-End
- Indexing
- Extended Operations
- ACLs
- Custom Objects
- Replication

Criticisms

- Isn't there just another database engine below all of this?
  - Yes, but not necessarily relational.
  - OpenLDAP has optional back-ends.
  - Commercial LDAP uses custom databases.
- No way to do complex joins to grab joined datasets.
  - True. This is not optimal for this kind of work.
- Aren't wildcard queries and anonymous or lower-user binds just as insecure as SQL?
  - Letting reads slip less dangerous than rights.
  - Access rights solve everything!
- LDAP cannot match the load of SQL
  - Given the diversity of LDAP implementations, we know this is a bit of a strawman.
  - Obviously, optimized for different purposes.
  - Let's unpack this.
- Not optimized for binary data.
  - Neither is XML.
  - It all depends on function.


SUSE®
We adapt. You succeed.