LDAP: a view from the machine room

Felix von Leitner
CCC Berlin
felix-ldap@fefe.de

August 2003

Abstract

LDAP is the protocol behind all major directory services, whether they come from IBM, Microsoft of Novell. Most presentations about LDAP are very high level, talking about replication, LDAP schema, about the wonderful scalability of LDAP, things like that. This talk is a view from the machine room, it's about how the protocol works.

Agenda

1. What does an LDAP message look like?
2. ASN.1, BER and DER
3. How do ACLs work in LDAP?
4. How does authentication work in LDAP?
5. How does replication work in LDAP?
6. What is LDAP particularly good for?

First of all

- LDAP is a descendant of DAP
- DAP is the Directory Access Protocol from X.500
- X.500 is an extremely messy and bloated pile of garbage
- DAP uses the OSI protocol stack
- The main (if not only!) difference between DAP and LDAP is that LDAP uses TCP/IP instead
- The ”L” stands for lightweight, but there’s anything LDAP is not, it’s lightweight

What does an LDAP conversation look like?

- Client opens TCP connection to server, port 389, and writes *Bind Request*
- Server writes *Bind Response*
- Client writes *Search Request*
- Server writes n *Search Result Entries*
- Server writes *Search Result Done*
- Client writes *Unbind Request*
- Server drops TCP connection

What does an LDAP message look like?

Here is a BindRequest:

```
0^L^B^A^A‘^G^B^A^C^D^@<80>^@
```

Yep, it’s binary.

It doesn’t look binary in the LDAP RFCs, thought.

That’s because the LDAP RFCs are pretty much worthless.

Huh?

LDAP is specified in ASN.1, which is a meta language to describe a language. It turns out that ASN.1 is pretty much worthless as well, because there are many different ASN.1 encodings, which turn out to look completely different on the wire.

However, most ASN.1 users use either BER or DER encoding. Those are actually the same encoding, but BER allows the wire equivalent of writing ”0023” when you mean ”23”.

ASN.1

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                  -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }
```

BER and DER

Here's how you format a 32-bit integer in BER.

In BER and DER you always write a tag first, then the length, then the actual value.

The tag looks like this:

```
+--------+-------+--------+
| class  | type  | tag    |
| 2 bits | 1 bit | 5 bits |
+--------+-------+--------+
```

BER and DER

The tag class is an enum. The values are:

0: universal (stuff like integer, boolean, string)

1: application (same, but the tag is application defined)

2: private (similar to application)

3: context specific (not used in LDAP)

BER and DER – application and private

Application tags are used in LDAP to specify what kind of message it is.

The LDAP spec says that each BindRequest begins with a SEQUENCE. So you don't actually have to tell the other side it's a sequence, as long as you say that it's a BindRequest.

The LDAP specs say BindRequest is an Application 0, so you write a sequence except that the ASN.1 tag does not say "universal, sequence" but it says "application, 0".

BER and DER

The tag type is a bit that can be PRIMITIVE (0) or CONSTRUCTED (1).

Integers, strings and booleans are primitive, sequences (think `struct`) and sets (think array) are constructed.

The length of the object is counted in bytes. However, the protocol should not limit the length arbitrarily, so BER encoding has this clever bit twiddling encoding:

1. if the length is smaller than 128, write the length in one byte

2. otherwise, write the length of the length in bytes plus 128 as one byte, then all the needed bytes of the length.

BER and DER

Yes, you read that right: you don't even know the size of the length in advance!

You might say: let's be lazy and not fiddle bits and always write the length as 4 bytes, there won't be objects larger than 4 GB in an LDAP directory.

You would be right, but this is only legal in BER, not in DER. LDAP uses DER.

By the way: BER == "Basic Encoding Rules".

DER == "Distinguished Encoding Rules".

Do you feel distinguished already?

BER and DER

If you are an experienced programmer you already know the kind of bugs that other people's LDAP implementations will have.

What if you encode 5 bytes in your LDAP Bind Request and the protocol parser only has 32-bit integers and does not check for overflow?

What if the protocol parser uses int instead of unsigned int?

These bugs were already found in various LDAP implementations.

But wait! There's more!

BER and DER

I haven't told you yet how BER and DER encode the actual integer.

You guessed it: variable length. You encode as many bytes as needed by the integer, big endian.

ASN.1 makes no difference between signed and unsigned integers.

So, if you want to write an unsigned integer (all integers in LDAP itself are unsigned, by the way), how does the other side know whether the other bytes in the integer need to be set to 0x00 (positive) or 0xff (negative)?

The answer is: you have to make sure to encode at least one leading zero bit. So, to encode 32767, you write three bytes in DER: 0x00, 0xff, 0xff.

BER and DER

To put it all together, this is how we encode the integer 23 in DER:

(tag: universal, primitive, integer) 0x00

(length) 0x01

(23) 0x17

This does not look very wasteful, but a boolean is basically an integer that is either 0 or 1. To encode a one bit value in DER, you write three bytes (24 bits).

On the other hand, the encoding makes it possible to skip over parts you don't understand or implement.

What's wrong with this encoding?

Encoding a string is straightforward. You call strlen, with that you calculate how many bytes you need to represent the length (usually 1), then you add one for the tag, and you have the size requirement.

But LDAP uses sequences with sequences inside, with integers and strings inside. So the know how much space you will need to the encoded length of the outer sequence, you will need to find the length requirements of the inner sequences, and for that you need to look at all the strings and integers and calculate the size requirements for them.

Then you know the size requirement of the outer sequence. To write the inner sequence, you need to iterate over all the inner data again.

What's wrong with this encoding?

If you write the inner sequences first, to save calculating the length again and again, you will need to copy them around into the final buffer again. This trashes the cache and would not be necessary if LDAP used a less stoned encoding.

Opportunities for bugs

I just told you that the encoder is needlessly complex.

What about the decoder?

There are lots of opportunities to generate invalid encoded strings.

For each inner string and integer, the parser much make sure the end of that object does not cross past the length specified in any of the outer sequences or sets. You end up carrying around a lot of state.

Let's decipher that LDAP BindRequest

`0^L^B^A^A'^G^B^A^C^D^@<80>^@`

The `0^L` indicates a SEQUENCE of length 12.

The `^B^A^A` is the integer 1 (that is the LDAP message ID).

The `'^G` indicates an (APPLICATION 0) SEQUENCE of length 7.

The `^B^A^C` is the integer 3 (LDAP version 3).

The `^D^@` is a string of length 0 (no user name, i.e. anonymous bind).

The `<0x80>^@` is a string of length 0 with private tag 0. The private tag says it's the first choice in a list of possibilities, the simple bind. LDAP also specifies SASL authentication, with private tag 3.

What's this message ID business?

LDAP operates like IMAP (in case you know the IMAP protocol).

Each request gets a message number.

The server then answers them in the order it likes most, and says which answer is for which message number.

LDAP Search Requests

LDAP search requests have these parameters:

- base object (string: ”o=fefe,c=de”)
- scope (enum: ”base object”, ”single level”, ”whole subtree”)
- deref (enum, to what extent should server follow aliases)
- size limit, time limit, whether to skip the values (only return keys)
- a filter

LDAP search request filters

- AND of filters, OR of filters, NOT a filter
- attribute value assertion: $=$, $\geq$, $\leq$
- substring match (prefix, suffix, any)
- attribute present (”has an email address”)
- approximate match (not actually specified)

LDAP search requests: scope

LDAP is meant to be hierarchical.

LDAP does not actually specify the hierarchy, but typical examples are

```
o=fefe,c=de
```

or

```
uid=felix-ldap,dc=fefe,dc=de
```

In the first case, having a base object of "`c=de`" will find everyone in Germany.

LDAP search requests: schema

The OpenLDAP people keep on talking about LDAP schema definitions and stuff like that, but the truth is: there is no spoo^Wschema.

A schema would be some external notion on what record formats to expect. The server can then reject non-conforming directory entries.

But what should the client do if the server answers with a non-conforming directory entry? That's why the whole schema idea is basically bogus.

LDAP should be viewed as something like SQL: it's a way to find records in a table in a database. Except that there is only one table in the database. And that hierarchy notion may help you narrow down what you were looking for.

What LDAP can handle

LDAP can basically handle records with arbitrary combinations of keys and values.

The LDAP protocol can handle the absence of an attribute, or there can be more than one occurrance of the same key in a record. That makes it quite flexible – have two email addresses? Just add one more `mail` attribute!

If you forget about aliases, referrals, that schema bullshit, SSL and SASL, LDAP is actually quite lightweight. But don't tell anyone I said that!

Other LDAP requests

LDAP also has:

1. ModifyRequests (add, delete and replace attributes in a record)
2. AddRequest (add a record)
3. DelRequest (remove a record)
4. ModifyDNRequest (change DN of a record)
5. CompareRequest (completely worthless, forget that I mentioned them)

What's this DN stuff?

LDAP is mostly like an SQL database, except that there is only one table, and there is only one key which must be unique.

This unique key is called DN, or Distinguished Name.

It looks like this: "cn=Fefe,ou=Berlin,o=ccc,c=de". You can put spaces after the commas if you want. It is a good idea (but not required by the protocol) that this record also contains attributes "ou=Berlin" and "cn=Fefe".

If you want to modify a record, you specify it by DN. That's why you need a special ModifyDNRequest to modify the DN itself.

ACLs

Did you see me mention any ACLs?

No?

That's because there are no ACLS in the protocol or specs. If a server wants to offer ACLs, it has to invent a way to do it itself.

This makes switching LDAP servers a royal pain in the ass.

It has also lead to the worst ACL implementation to ever walk the earth: OpenLDAPs.

What is so bad about OpenLDAP's ACLs?

OpenLDAP ACLs look like this:

```
access to dn=".*o=fefe,c=de" attr=userPassword
       by self write
       by anonymous auth
       by group="cn=Admins,ou=admins,o=fefe,c=de" write
       by * none
```

Granting write access implies granting read access.

Doesn't look so bad? Think again!

What is so bad about OpenLDAP's ACLs?

ACLs are sorted by DN to which they apply. You can't say "everyone can write his own record", you have to say that in each ACL.

The DN is matched first. So to check whether you can read my email address, OpenLDAP will basically check EVERY special case ACL in the ACL list, even the ACLs that only apply to granting write access, because those imply read access.

Typical ACL lists contain something like 3 ACLs to specify who can read what, and something like 50 ACLs to specify who can modify what. The result is that the performance of writable OpenLDAP servers is **really** bad.

What is so bad about OpenLDAP's ACLs?

The solution is to have two OpenLDAP servers. One with writing enabled, one read only. The read only one is a slave of the write server and gets auto updated.

The read server is quick because it only has 3 ACLs to check.

The write server is slow like molasses, but it doesn't matter because basically only the admins and users changing stuff will have to suffer. All the automated read accesses are quick. Well, relatively quick.

It's OK to run both servers on the same single CPU box, on different ports.

What else sucks about OpenLDAP?

The default storage backend is Berkeley DB, the worst piece of crapware under the sun. People regularly lose data and get corrupted directories because Berkeley DB simply sucks.

"Of course it doesn't work but look how fast it is!"

OpenLDAP can also use GDBM as back-end, but for that you have to enable the LDBM backend (at compile time!), disable Berkeley DB, and link against GDBM. This is slower but at least it works.

Summary

LDAP itself is surprisingly harmless.

It's the LDAP specs that suck.

And the ASN.1 encodings.

And the typical implementations. OpenLDAP is bad, but most of the commercial offerings suck even worse, many are even based on old versions of OpenLDAP!

LDAP itself can be implemented in 24k (statically linked, a read-only server), and a trivial LDAP client (in this case, a `checkpassword` can be done in about 10k (again, statically linked).

Join me!

My LDAP project can be found at `http://www.fefe.de/tinyldap/`.

My LDAP server is quite small, and it can handle a thousand queries per second on my notebook while forking a new client and a new server process for each request. And each request opens a new TCP connection.

So obviously LDAP can be done small and efficiently. It's just that nobody has done it before me.

Tinyldap still needs write support. I have spent some time thinking about this, but decided to postpone the implementation until someone pays for it. Many people are enduring terrible pain with OpenLDAP. I think it's a question of time.

Email me at `felix-ldap@fefe.de`!