BRENDAN QUINN

lessons learned from living with LDAP

Brendan Quinn has more than 14 years of experience as a sysadmin, security engineer, and infrastrucure engineer. He is currently a Senior Infrastructure Project Engineer at London Business School. He is also a musician and audio engineer.

brendan.quinn@gmail.com

LDAP CAN BE A DOUBLE-EDGED SWORD. On the one hand, you have data available on the network, which is easy to access in a standard way. On the other, you have a system that doesn't work or behave like a normal database. At London Business School, we've had a central LDAP repository for years now, and over time it's come to be at the center of our network. We have applications and network devices that use LDAP for authentication and authorization. We have applications that depend on data in the LDAP repository for core pieces of their functionality. We're even using LDAP to route every piece of email that enters our network. Over the past few years, I've learned a few things the hard way about LDAP integration and performance. I'd like to share a few of those lessons, in the hope of saving you some headaches.

"Why Is the Web Site Slow?" or, Thinking About Performance Tuning

Performance can be a serious problem when dealing with LDAP. There are lots of documents out there that explain the mechanics of tuning particular LDAP servers, and I've included a few links to get you started [1, 2]. In the Sun Directory Server, performance tuning consists of building indexes and adjusting cache sizes. Tuning OpenLDAP while using the Berkeley DB backend will be similar. However, it's difficult to tune an LDAP server unless you understand the traffic it's likely to see. So before you dive in and start tweaking, start by asking a few questions.

Is there a particular application or applications that generate large numbers of queries? If so, can you predict how many?

At our site, we route all the email for our 5000+ person userbase using data stored in LDAP [3]. As you might imagine, this generates quite a lot of queries. We can estimate the number of queries the LDAP server will get based on the number of queries needed to route each individual mail message, multiplied by the number of mail messages handled per day. Looking at the mail logs, we can also get the number of messages processed per minute during peak times or mail floods. In this

case, we tested the LDAP architecture by modeling significantly more traffic than this amount of mail would generate. When we were happy with the response times of the LDAP server under the modeled load, we knew that the gating factor for mail performance would be the underlying mail architecture and not LDAP.

What filters are being used? Does the application use programmatically generated complex filters?

The kinds of filters used by directory-enabled applications can often be more complex and slower than you might expect based on the kinds of data being returned. For example, we have one application that always appends a * to the filters it generates, even though it always knows the exact data it's searching for. So although I'd expected an exact match filter, in fact it always used a substring filter. In practice, this meant that the exact match indexes I'd built were useless; I needed to build substring indexes instead. If you're not sure which filters are being used, check the LDAP server access logs. Logs don't lie, and you can usually see exactly which filters the applications are using.

Once you understand the kinds of filters your applications are using and you have some estimates of what kind of traffic your LDAP architecture is expected to handle, you can start tuning your LDAP servers.

I've written a couple of tools in Perl that you may find useful for testing LDAP server performance [4, 5]. They depend on Net::LDAP.

"Why can't I change my address?" or, A Brief Diversion into Objects, Namespace, and Access Control

In the next few sections I'll be talking a lot about how applications use LDAP. Before I do, let's discuss a few key concepts. For a much more complete explanation, it's worth reading Tim Howes, Mark Smith, and Gordon Good's excellent and comprehensive book [6].

LDAP is object-oriented. That's a bit of a confusing term in the context of a data store, because traditional object orientation is about tightly binding data and logic. LDAP objects don't have logic; they consist purely of data. What object orientation actually means in this context is that data in LDAP is structured as objects, which contain attributes and belong to object classes. Attributes name the bits of data and define the syntax of that data. Object classes define what attributes an object can (and must) have. Object classes use inheritance, which just means that they inherit attributes from their parent object classes.

The namespace of a repository is the structure of that repository. LDAP uses a hierarchical data model. Translated, that means that data is stored and accessed in LDAP as objects hanging in a tree. Tree nodes are themselves objects. Every object has an address, or Distinguished Name (DN), which consists of the list of tree nodes that must be followed to get to the object (see Figure 1).

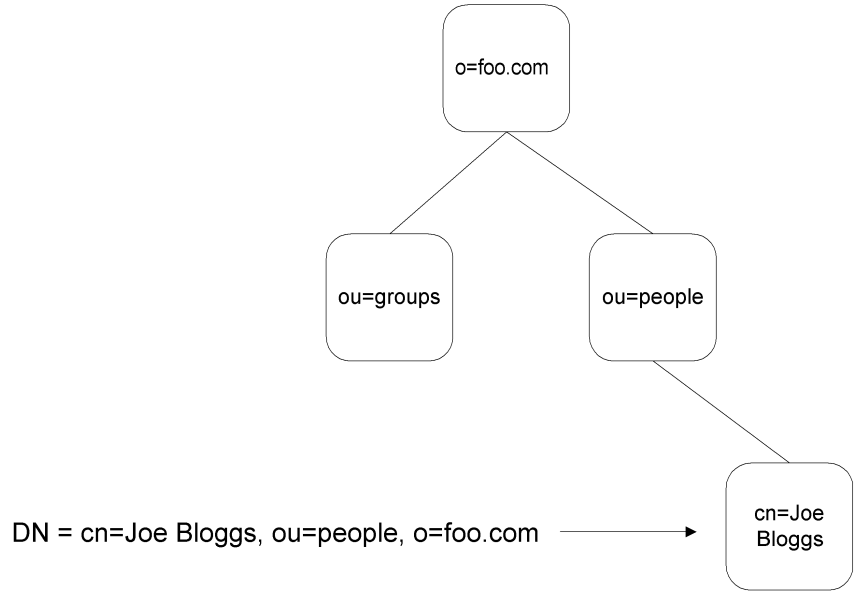

FIGURE 1

The most common way of implementing access control on an LDAP repository is through the use of Access Control Instructions (ACIs). ACIs generally use a syntax similar to that used for LDAP searches (generally called filters). The syntax does differ from vendor to vendor though, so you'll have to look at the documentation for your particular LDAP server for the specifics. The nice thing about ACIs is that they allow you to manage your access control rules in the LDAP repository itself, by setting the "aci" attribute on any object in the tree, nodes and leaves alike.

ACIs also allow the LDAP server to make use of all of the data contained in the LDAP repository when making access control decisions. If the client has provided credentials for an object contained in the repository (called binding), the server can perform internal lookups for group membership, attribute values, and so on.

"But it says on the Web site that it's LDAP integrated . . .", or, Using LDAP for Authentication and Authorization

So you've bought an application or a piece of network equipment, and you want it to use LDAP for authentication and authorization. The vendor claims, "Easy integration with your LDAP repository!" Unfortunately, every manufacturer seems to have a slightly different meaning for this statement.

In general, the Authentication-Authorization (AA) protocol should look something like this:

1. The application performs an anonymous bind against the LDAP server.
2. The application does a search (generally returning only the Distinguished Name) to verify that the username provided matches an object existing in the repository.
3. The application attempts to bind as the DN returned by the previous search, using the password provided.
4. The application, now bound to LDAP as the user attempting authentication, searches for particular attributes in the user's object.
5. The application checks the value of the returned attribute to determine whether the authenticated user is authorized to access the application.

The semantics of the AA process will differ slightly from vendor to vendor, but the overall approach should be fairly similar to that listed here. Unfortunately, there are a few things that can go wrong even with this simple case.

First, anonymous searches (even for DNs) are sometimes forbidden for security reasons, especially in cases where the LDAP repository must be exposed to the open Internet. This will cause the application always to fail to authenticate, because it will be unable to determine which DN to attempt to bind with when validating the user's password. In some cases, the application can be configured with credentials to use for the DN search, rather than using an anonymous bind. Other applications can be configured to construct the DN from the username provided, although this requires that the namespace use the username as part of the DN, which won't always be the case. If the application doesn't support either of these approaches, you'll have to allow anonymous searches for DNs, or not use the application at all. If your site doesn't allow anonymous searches for DNs, it's important to ask the vendor whether the application supports one of these alternatives before purchase.

Second, at many sites, users are forbidden from reading some or all of their own attributes. This will cause authentication to succeed but authorization to fail. The simplest way of dealing with this is to permit the users to read any attributes in their own object that are used for authorization, but this will not always be permitted by the security policies. Some applications will support binding as another user (typically the one used for the DN search) to search the user object for the required attributes.

Finally, which attributes does the application use for authorization? Some applications will allow you to specify an attribute name to check and values to look for. Some will allow you to specify which attribute to check but will require a fixed, vendor-specified syntax to match against. Still others will require that you add a specific attribute or attributes to your schema, generally with vendor-specified syntax. Whatever the case, you'll need to think about where the data stored in these attributes comes from, and how it's maintained and managed.

It's worth noting that some applications that claim to support LDAP authentication don't use the generic process outlined here. In some cases they're designed to bind as an administrative user (with full access rights), retrieve the uid and userpassword attributes, and attempt to validate the passwords themselves. This is poor practice, and you should avoid any vendor who uses such a brain-dead approach.

"I know, we'll just use LDAP for everything!" or, Directory-Enabled Applications and You

The most common use of LDAP is for authentication and authorization, but there are also applications, such as our email system, that make operational decisions based on the data contained in the repository. This class of directory-enabled applications generally makes use of application-specific schema (although possibly making extensive use of one of the standard schema as well). An application of this type will often expect to manage the attributes in the custom schema itself, writing directly to the repository. As many sites don't permit users to modify most of their attributes directly, this generally requires giving the application some kind of administrative access. At our site, we've handled this by setting up a number of

group objects (in ou=Groups), which we reference in the ACIs on different parts of the repository. When an application requires administrative access, we create a user object specific to that application, and we add that user's DN to the group object's uniqueMember attribute. This minimizes the need for rewriting ACIs and brings the number of ACIs needed down to a manageable level.

When you're dealing with applications that perform LDAP writes as well as reads, it's important to make sure that the application developers understand that an LDAP repository is not a traditional database. LDAP servers are highly optimized for reads over writes, and write operations can take a fair bit of time. In extreme cases, too many write operations can slow an LDAP server down to a crawl. For example, at one point we had an application developer who was using LDAP to store session cache information. This was generating an LDAP write for every click that every user made on his or her application's Web site. The solution in that case was to gently explain to the developer why this was such a bad idea. You're less likely to run into this issue with commercial applications, but when applications are being developed in-house it's worth repeating the "write rarely, read often" principle at every opportunity.

One more thing on writes is worth mentioning, before we move on. If you're using a Master/Slave LDAP architecture, when a client attempts a write to the Slave, it will get a referral to the Master. In my experience, most applications only allow you to configure a single LDAP server pool, which is used for both reads and writes. If the application needs to write to LDAP and supports LDAP referrals, you should configure it to use Slaves. If the application doesn't support LDAP referrals, you will have to configure it to use the Master. Most applications that claim to support LDAP referrals work as advertised, but, not surprisingly, there are a few that fail in unexpected ways. If your site uses LDAP referrals and you're having problems with a particular application, try configuring it to use the Master instead. If this fixes the problem, you'll know that the failure most likely is in the way the application handles LDAP referrals.

"But that data was in the repository yesterday . . . ," or, Issues Arising from Data Management

The data in your LDAP repository has to come from somewhere. Some data will originate in LDAP and be maintained directly there. Some data will likely originate in one or more administrative databases. Some may originate in another, special-purpose LDAP server, such as Microsoft Active Directory. All of this data will need to be synchronized. When there are applications making decisions based on LDAP data, synchronization issues can have unexpected consequences.

Let's examine a hypothetical site, where most of the data in ou=people originates in a single administrative database, while passwords are managed directly in the LDAP repository. Data from this database is synchronized with LDAP by using a process that runs once per day. One day, somebody working with the database makes a mistake and accidentally deletes the entire marketing department. This is noticed relatively quickly (within an hour or so), and the database is corrected. Unfortunately, between the deletion and the correction, the daily LDAP update has run. This resulted in all of the users in the marketing department having their user objects deleted. The LDAP administrator doesn't find out about the issue until the call comes in from the head of marketing, who is angry that

no one in the department can get onto the network. Our poor LDAP admin is faced with a quandary. If the database synchronization process is rerun, all the users will be recreated, but they will all need their passwords reset, and some of their usernames may have changed. If the repository is restored by using the last backup, those in the rest of the company who changed their password since that backup will have their password set back to before the change.

What could have been done to prevent this from happening in the first place? You can't completely prevent human error, so there's nothing that could have prevented the initial deletion from the database. However, this type of failure could have been anticipated and planned for. For example, the synchronization process could have been designed to be less automatic, requiring that someone look at an analysis of the changes that will be made and start the process manually. Alternatively, the synchronization process itself might have had a sanity check built in: Build in some logic that would notice that the process was being asked to delete a large number of user objects, and refuse to proceed without human intervention and approval. Did the objects need to be deleted immediately at all? What if the synchronization process had instead marked the object as inactive, say, by prepending a date string to the hash contained in the userpassword attribute with the date that the account had been made inactive? The process could then automatically delete inactive user objects after a specified time period. If you needed to recover the password you could simply strip out the date string.

Making your synchronization processes smarter will help, but not all data problems will be as obvious as this. Bad data is everywhere and isn't going to go away anytime soon. The best that you can do is to try to understand the most likely sources of bad data, and try to minimize the impact of any predictable failure conditions.

One way that LDAP differs from traditional databases is that the data isn't designed for one particular purpose. Once data is in LDAP, people will find new ways of making use of it that can't be anticipated. This is the very thing that makes LDAP so powerful. Unfortunately, this complicates the job of data management. Each application and data source will have a different conception about the purpose of the data in the repository. Understanding what applications are accessing the repository and how they're using the data is an essential step in understanding how to manage the data that resides there.

"conn=37288428 op=81303 RESULT", or, Conclusion

You can tell that the LDAP infrastructure is working well when it's become just like the plumbing. Everyone uses it, but nobody ever thinks about it. Unfortunately, when the LDAP infrastructure isn't working well, it's more like public transportation. Everyone still uses it, but they complain about it all the time. With some thought and attention, an LDAP repository can be one of the most reliable parts of your network infrastructure. I hope this article has given you a few things to think about along the way.

REFERENCES

[1] Steve Lopez, "Solaris Operating Environment LDAP Capacity Planning and Performance Tuning," is a very comprehensive document, which, although primarily focused on Solaris and the Sun LDAP server, provides a

wealth of information useful in the general case. Available at http://www.sun.com/blueprints/0502/816-4829-10.pdf.

[2] "OpenLDAP FAQ-O-Matic: Performance Tuning" provides an overview of performance tuning OpenLDAP. Available at http://www.openldap.org/faq/data/cache/190.html.

[3] Brendan Quinn, "Integrating Exim with LDAP for Mail Relaying—A Case Study." Available at http://www.uit.co.uk/exim-conference/full-papers/brendan-quinn.pdf.

[4] lookup.pl: This is basically a simple LDAP search tool, but with all the search options pushed into a configuration file, and with high-resolution timing metrics added. The advantage of this is that it lets you save models of different types of searches. Available at http://phd.london.edu/bquinn/lookup/.

[5] ldap-load.pl: This is an LDAP load testing tool based on lookup.pl. It simulates load by spawning processes, which then send queries to the LDAP server. It's mostly useful for generating peak loads, but it can be used to generate sustained load as well. Available at http://phd.london.edu/bquinn/ldap-load/.

[6] Timothy A. Howes, Mark C. Smith, and Gordon S. Good, *Understanding and Deploying LDAP Directory Services* (Macmillan Technical Publishing, 1998).