computers are bad

You are receiving this facsimile because you signed up for fax delivery of this newsletter. To stop delivery, contact Computers Are Bad by email or fax.

https://computer.rip - me@computer.rip - fax: +1 (505) 926-5492

2021-05-10 lightweight as in ldap

Programming note: I have posted two videos to my poorly-tended YouTube account. They are part two of the video about Manzano base, and a rough version of a conference presentation on security of aviation radionavigaton technologies.

I've mentioned LDAP several times as of late. Most recently, when I said I would write about it. And here we are! I will not provide a complete or thorough explanation of LDAP because doing so would easily fill a book, and I'm not sure that I'm prepared to be the kind of person who has written a book on LDAP. But I will try to give you a general understanding of what LDAP is, how it works, and why it is such a monumental pain in the ass.

I've also mentioned it, though, in the context of the OSI protocols. This is because LDAP is a direct descendent of one of the great visions of the OSI project: a grand, unified directory infrastructure with global addressability and integration with the other OSI protocols. This is an example of the ambition and failure of the OSI concept: in practice, directory services have proven to be fairly special-purpose, limited to enterprise environments, and intentionally limited in scope (e.g. kept internal for security reasons). OSI contemplated a directory infrastructure which was basically the opposite in every regard. It did not survive to the modern age, except in various bits and pieces which are still widely used in... once again, crypto infrastructure. Common crypto certificate formats are ASN.1 serialized (as we mentioned last week) because they are from the OSI directory service, X.500.

Before we get into the weeds, though, let's understand the high level objectives. What even is a directory, or a directory service?

It's a digital telephone directory.

This answer is so simple and naive that it almost cannot be true, and yet it is. Remember that the whole OSI deal was in many ways a product of the telephone industry, and that the telephone industry has always favored more complex, powerful, integrated solutions over simpler, independent, but composable solutions. One thing the telephone industry knew well, and had a surprisingly sophisticated approach to, was the white pages.

If you think about it, the humble telephone directory was a surprisingly central component of the bureaucracy of the typical 1970s enterprise. Today, historians often review archived institutional and corporate telephone directories as a way to figure out the timelines of historical figures. Corporate histories often use the telephone directory as a main organizing source, since it documents both the changing staff and

the changing structure of the organization (traditional corporate directories often had an org chart in the front pages to boot!).

Across the many functional areas of a business, the telephone directory was a unifying source of truth--or authority--for the structure and membership of the organization. For consumer telephone service, directories had a less complex structure but were an undertaking in their own way due to the sheer number of subscribers. Telephone providers put computers to work at the job of collecting, sorting, and printing their subscriber's directory entries very early on. The information in the published white pages was an excerpt or report from the company's subscriber rolls, and so was closely tied to other important functions like billing and service management.

Inside the industry, the directory referred to all this and more: the unified, authoritative information on the users of the system.

This concept was extended to the world of computing in the form of X.500 and its accompanying OSI network protocols for access to X.500 information. At its root, LDAP is an alternative protocol to access X.500, and so there are substantial similarities between X.500 and the X.500-like substance that we now refer to as an LDAP server. In fact, there is no such thing as an "LDAP server" in the sense that LDAP remains a protocol to access an X.500 compliant directory, but in practice LDAP is now usually used with backends that were designed specifically for LDAP and avoid much of the complexity of X.500 in the sense of the OSI model. The situation today is such that "X.500" and "LDAP" are closely related concepts which are difficult to fully untangle; X.500 is very much alive and well if you accept the caveat that it is only used in the constrained form of corporate directories accessed by alternative methods [1].

The basic structure of X.500 is called the Directory Information Tree, or DIT. The DIT is a hierarchical database which stores *objects* that possess *attributes,* which are basically key-value pairs belonging to the object. Objects can be queried for based on their attributes, using a form called the Distinguished Name. DNs are made up of a set of attributes which uniquely identify an object at each level of the hierarchy. For example, an idealized X.500 DN, in the same notation as used by LDAP/LDIF (notation for DNs varies by X.500 protocol), looks like this: cn=J. B. Crawford,ou=Blogger,o=Seventh Standard,c=us. This DN identifies an object by, from top to bottom, country, organization, organizational unit, and common name. Common name is an attribute which contains a human-readable name for the object and is, conventionally, widely used for the identification of that object.

Note some things about this concept: first, the structure is rooted in the US. How does the namespace work, exactly? Who determines organizations under countries? Originally, X.500 was intended to be operated much like DNS, as a distributed system of many servers operating a shared namespace. Space in that namespace would be managed through a registry, which would be SRI or Network Solutions or whatever [2].

Second, this whole concept of identifying objects by attributes seems like it's very subject to conventions. It is, but you must resist the urge to hear "hierarchical store of objects with attributes" and think of X.500 as being a lightly-structured, flexible data store like a modern "NoSQL." In reality it is not, X.500 is highly structured through the use of *schemas.*

We mostly use the term "schema" when talking about relational databases or markup languages. X.500 schemas serve the same function of describing the *structure* of objects in the DIT but look and feel different because they are highly object-oriented. That is, an X.500 schema is made up of classes. Classes can be

inherited from other classes, in which case their attributes are merged. Resultingly there is not only a hierarchy of data, but of types. Objects can be instances of multiple classes, in which case they must provide the attributes of all of those classes, which may overlap. It's seemingly simple but can get confusing very fast.

Let's illustrate this by taking a look at a common X.500 class: organizationalPerson, or 2.5.6.7. What's up with that number? Remember the whole snmp thing? Yes, X.500 makes use of OIDs to, among other things, identify classes. That said, we commonly (and especially in the case of LDAP) deal only with their names.

While organizationalPerson does not require any attributes, it suggests things like these:

- title
- telexNumber
- telephoneNumber
- postalAddress
- physicalDeliveryOfficeName

You will notice that this list is dated, and missing obvious things like name. The former is because it is in fact very old, the latter is because organizationalPerson is an auxiliary class and so is intended to be applied to objects only in addition to other classes. Namely, organizationalPerson is usually applied to objects alongside Person, which has some basics like:

- cn (common name, required)
- sn (surname, required)
- givenName
- telephoneNumber
- assistant (as in, reference to this person's assistant)

You will notice that this class both overlaps with organizationalPerson on telephoneNumber, but also has some odd things like assistant that seem to be specific to an organization. Why the two different classes, then? Conway observed that the structure of systems resembles the structure of their creators; X.500 is no exception. organizationalPerson was written more as part of an effort to represent organizations than as part of an effort to represent people, these two efforts were not as well harmonized as you would hope for.

An object has a "primary" or "core" type. This is referred to as its structural class, and the class itself must be specially marked as structural. This is important for several reasons that are mostly under the hood of the X.500 implementation, but it's useful to know that Person is a structural class... so an X.500 entry representing a human being should have a core type of Person, but in most cases will have multiple auxiliary types bolted on to provide additional information.

That's a lot about the conceptual design of X.500... or really just the core concept of the data structure, ignoring basically the entire transactional concept which is more complicated than you could ever imagine. It's enough to get more into LDAP, though.

Before we go fully into the LDAPverse, though, it's useful to understand how LDAP is really used. This swerves right from OSI to one of my other favorite topics, Network Operating Systems [3].

For a group of computers to act like a unified computing environment, they must have a central concept of a user. This is most often thought of in the context of authentication and authorization, but a user directory is also necessary to enable features like messaging. Further, the user directory itself (e.g. the ability to use the computer as a telephone directory) is considered a feature of a network computing environment in its own right.

In almost all network computing environments, this user directory is descended from X.500. This is seen in the form of Microsoft Active Directory for Windows (modern Windows does not actually use LDAP to interact with the AD domain controller, but instead a different directory access implementation called NT LAN Manager or NTLM), and LDAP for Linux and MacOS (we will not discuss NIS for Linux now, but perhaps in the future).

In these systems, the directory server acts as the source of basic information on the user. Consider another important LDAP class, PosixAccount. PosixAccount adds attributes like uid, homeDirectory, and gecos that reflect the user account metadata expected by POSIX [4]. It is possible to perform authentication against LDAP as well, but it comes with limitations and security concerns that make it uncommon in practice for operating systems. Both Windows and Unix-like environments now generally use Kerberos for authentication.

Many things have changed in the transition from the grand vision of X.500 to the reality of LDAP for information on user accounts. First, the concept of a single unified X.500 namespace has been wholly abandoned. It's complex to implement, and it's not clear that it's something anyone ever wanted, anyway, as federation of directories between organizations brings significant security and compliance concerns.

Instead, modern directories usually use DNS as their root organizational hierarchy. This basically involves cramming shim objects into the DIT that reflect the DNS hierarchy. The example DN I mentioned earlier would more often be seen today as cn=J. B. Crawford,dc=computer,dc=rip. dc here is Domain Component, and domain components are represented in the same order as in DNS because LDAP uses the same confused right-to-left hierarchical representation (AD does it the correct way around).

Another major change has been to the structure. The original intention was that the X.500 hierarchy should represent the structure of the organization. This is uncommon today, because it introduced a maintenance headache (moving objects around the directory as people changed positions) and didn't have a lot of advantages in practice. Instead LDAP objects are more commonly grouped by their high-level purpose. For example, user accounts are often placed in an OU called "accounts" or "users." All in all, this marks a more general trend that LDAP has become a system only for software consumption, and there is minimal concern today about LDAP being browseable by human users.

So let's consider some details of how LDAP works. First off, LDAP is a binary protocol that uses a representation based on ASN.1. That said, LDAP is almost always used with LDAP Data Interchange Format, or LDIF, which is a textual representation. So it's very common to talk about LDAP "data" and "objects" in LDIF format, but understand that LDIF is just a user aid and is not how LDAP data is represented "in actuality."

LDAP provides more or less the verbs you would expect: ADD, DELETE, MODIFY. These are not especially interesting. The SEARCH operation, however, is where much of the in-use complexity of LDAP resides. SEARCH is a general-purpose verb to retrieve

information from an LDAP DIT, and it is built to be very flexible. At its simplest, SEARCH can be invoked with a baseObject (a DN) and a scope of BaseObject, which just causes the server to return exactly the object identified by the DN.

In a more complex application, SEARCH can be invoked with a base path representing a subtree, a scope of wholeSubtree (means what it says), and a *filter.* The filter is a prefix-notation conditional statement that is applied to each candidate object; objects are only returned if the filter evaluates to true.

We can put these SEARCH concepts together into a very common LDAP SEARCH application, which is locating a user in a directory. A common configuration for a piece of software using LDAP for authentication would be:

baseObject: ou=users,dc=computer,dc=rip

scope: singleLevel

filter: (&(objectClass=PosixAccount)(uid=$user))

The $user here is a substitution tag which will be replaced by the user's username. Confusingly, in the PosixAccount class, uid refers to the user *name* while uidNumber is the value we usually refer to as uid.

A real headache comes about with groups. In authorization applications like RBAC, you commonly want to get the list of groups a user is a member of to make authorization decisions. There are multiple norms for representing groups in LDAP. Groups can have a list of accounts which are members, or accounts can have a list of groups they are a member of. Both are in common use, generally the former for Windows and the latter for UNIX-likes. This is where the flexibility of the filter expression becomes important: whatever "direction" the LDAP server represents the relationship, it's possible to go "the other way" by querying for the object type that contains the list with a filter expression that the list must contain the thing you're looking for. Because finding all users in a group is a less common requirement than finding all groups a user is in, a lot of LDAP clients in practice make somewhat narrow assumptions about how to find users but provide a more general (but also more irritating) configuration for finding group information [5].

Another complexity of LDAP in practice is authentication. A last important LDAP verb is BIND, BIND is used to assume the identity of a user in the directory. While anonymous access to LDAP is common, modern directory servers implement access control and limit access to sensitive values like password hashes to the users they belong to, for obvious reasons. This means that the formerly common approach of anonymously querying for a user to get their password hash and then checking the password should never be seen or heard of today. Instead, user authentication is done via BIND: the LDAP client attempts to BIND to the user (as an LDAP object) using the password provided by the user. If the server allows it, the user apparently provided the correct password. If the server doesn't allow it, the user better try again. In this way, the actual authentication method is the authentication method of the LDAP server itself [6].

There's a problem, though. Or rather, two. First, for security reasons, it's not necessarily a great idea to allow users to query for complete group information, and depending on how group membership is represented it is not necessarily practical to use access controls to allow a user to access only the group information they should know about. Second, applications often have a need to access directory information at

points other than when a user is actively logging in and the application has access to their password. For obvious reasons it is not a good idea for the application to store the user's password in plaintext for this purpose.

The solution is an irritating invention usually called a "manager." The manager is a non-person account (also called a system account) that an LDAP client uses in order to BIND to the LDAP server so that it is permitted to read information that is not available for anonymous query. Most commonly this is used for getting a user's group memberships. This is a particularly common setup because a lot of applications need access to user group information fairly frequently and do not strongly abstract their user information access, so they "cache" group information and update it from the LDAP server periodically--outside of the context of an authenticating user.

Very frequently this takes the form of periodically "synchronizing" the application's existing local user database with the LDAP server, a lazy bit of engineering that causes endless frustration for administrators but is also difficult to avoid as the reality is that the concepts of "user" and "group" simply vary far too widely between applications to completely centralize all user information in one place.

As mentioned earlier, all of the methods of authenticating against LDAP have appreciable limitations. For this reason, Kerberos is generally considered the superior authentication method and "real" LDAP authentication is not common at the OS level. That said, Kerberos configuration and clients are relatively complex, which is probably the main reason that many non-OS applications still use direct LDAP authentication.

In practice, directory servers are not usually set up as a standalone package. Usually they are one facet of a larger directory system or identity management system. Popular options are Microsoft Active Directory and Red Hat IDM (based on FreeIPA), but there are a number of other options out there. Each of these generally implement a directory service alongside a dedicated authentication service (usually Kerberos because it is powerful and well researched), a name service (DNS), and some type of policy engine. DNS might initially be surprising here, as it does not at first glance seem like a related concern. However, in practice, directory systems represent *device* just as much as people. Because each host needs to have a corresponding directory entry (particularly important with Kerberos where hosts need the ability to authenticate to other network services on their own), it's already necessary to maintain host information in the directory service which makes it a natural place to implement DNS. DHCP is also sometimes implemented as part of the directory service because there is overlap between the directory management functions and basic host management functions of DHCP, but this seems to be less common today because in enterprise orgs DHCP is more often part of an IPAM solution (e.g. Infoblox).

You might be surprised to hear that there are all of these inconsistencies and differences in LDAP implementations considering my claim that X.500 is strongly typed against schemas. The nature of this contradiction will be obvious to any DBA: for any non-trivial application, the schema will always be both too complex and not complex enough. The well-established X.500 and LDAP schemas, published for example in RFCs, don't have enough fields to express the full scope of information about users needed in any given application. Simultaneously, though, they provide so many types and attributes that there are multiple ways to solve a given problem. Any attempt to reduce one problem will inevitably make the other worse.

The long history of these systems only makes the problem more complicated, as there are multiple and sometimes conflicting historic schemas and approaches and it's hard

to get rid of any of them now. For this reason identity management solutions often come with some sort of “quick ref” documentation explaining the important aspects of the LDAP schema as they use it, to be used as an aid in configuring other LDAP clients.

I’m going to call this enough on the topic of LDAP for now... but there will be a followup coming. For me, this whole discussion of complex enterprise directory solutions raises a question: can we have the advantages of a directory service, namely a unified sense of identity, in a consumer environment?

The answer is yes, through the transformation of all software into a monthly subscription, but I want to talk a bit about the history of attempts at bringing the dream of the NOS to the home. Microsoft has tried at least a half dozen times and it has never really worked.

[1] As an example of this ontological complexity, Microsoft Active Directory is sometimes referred to as being an LDAP server or LDAP implementation. This is not true, but it’s also not untrue. It is perhaps more accurate to say that “Active Directory is an implementation of a modified form of X.500 which is commonly accessed using LDAP for interoperability” but that’s a mouthful and probably still not quite correct.

[2] Have I written about this here before? While IANA was long operated by Jon Postel who was famously benevolent, the function of ICANN was tossed around defense contractors for a while and then handed to Network Solutions, who turned out to be so comically evil that the power had to be taken away from them. ICANN didn’t turn out much better. It’s a whole story.

[3] Requisite explanatory footnote about network operating systems (NOS): the term has basically changed in definition midway through computer history. Today NOS generally refers to operating systems written for network appliances, like Cisco IOS. Up to the mid-’90s, though, it more commonly referred to a general-purpose operating system that was built specifically to be used as part of a network environment, such as Novell Netware. The salient features of NOS such as centralized user directories, inter-computer messaging, and shared access to storage and printers are present in all modern operating systems (sometimes with implementations borrowed from historic NOS) and so the use of the term NOS in this sense has faded away.

[4] This whole thing gets into some weird UNIX history, particularly the gecos and the aspect of LDAP’s UNIX-nerd cousin NIS. Maybe that’ll be a post some time.

[5] For how closely connected the concept of users and groups seems to be, this issue of the user->group query being irritatingly difficult is remarkably common in identity systems, even many modern “cloud” ones. Despite being a common requirement and one of the conceptually simpler options for authorization RBAC does not generally seem to be a first-class concern to the designers of directories.

[6] It’s possible to use a wide variety of network services for authentication in this way, by just passing the user’s credentials on and seeing if it works. I have seen a couple of web applications offer “IMAP authentication” in that way, presumably because small organizations are more likely to have central email than LDAP.