DBIS: Evolving NIS in LDAP

Author: Mark R. Bannister, September 2015

Background

Before I begin to write about DBIS, let me first explain my rationale, my motivation for entering this space, the reason I felt it was necessary to evolve RFC 2307. But first a history lesson. For those of you who don't know, RFC 2307 is an experimental protocol that was written in the late 1990s by Luke Howard that defined a way to represent NIS maps in LDAP. This catered for the following databases: *passwd*, *shadow*, *group*, *netgroup*, *services*, *protocols*, *rpc*, *hosts*, *ethers*, *networks* and *custom*. It enjoyed widespread adoption across the industry, despite its experimental nature. This was updated by the internet draft RFC 2307bis of which several versions were released between 2002 and 2009, although to this day RFC 2307bis remains a draft document only.

Shortly after the publication of RFC 2307, a number of different nss_ldap libraries were developed separately that implemented the protocol; the one developed by Sun Microsystems became the standard library in Solaris, and the one developed by Luke Howard became – and still is today – the defacto library on many Linux variants. These libraries would use the name service switch (NSS) framework to provide back-end functionality for C APIs that lookup name service information, such as getpwent() and gethostbyname(). If the information requested was not already cached by nscd (the name service caching daemon), the library would be responsible for connecting to an LDAP server to obtain the information.

This architectural model had some drawbacks. Firstly, not all of the maps were supported by NSS. I will give you two specific examples of this: while you could use the ypcat command to obtain data from custom NIS maps, there was no simple equivalent in the LDAP paradigm (you could use a ypcat wrapper, such as provided by my yp2ldap package on SourceForge, but that is an afterthought not a first class citizen). Also, the automounter, support for which was added by RFC 2307bis, has no NSS support either and so Linux autofs maintains its own LDAP configuration and manages it own LDAP connections leading to duplicated administrative and testing effort setting it all up.

Secondly, because the NSS libraries were responsible for opening their own LDAP connections, this would mean that potentially any process in the process table, owned by any user, may at any point need to open an LDAP connection. This leads to insurmountable complications when one must limit which users or which processes should be allowed to open

network ports or to communicate with LDAP services, and complicates the management of Kerberized LDAP connections. It also led to inefficiencies as multiple applications could instigate lookups for the same piece of data, and in troubleshooting problems as there was no central log of lookup requests that were being processed.

Part of this problem had attempted to be addressed in the informational RFC 4876 published in 2007, which provided a standard way to configure knowledge of which LDAP servers to use, what authentication scheme to use, search base, scope, attribute and object class remapping rules etc. However, it did not tackle the problem that a common library was initiating LDAP search operations, rather than a broker.

In 2005, when Sun Microsystems released Solaris 10, they used a new model where the NSS library did not perform the LDAP lookups, but instead it was lightweight and sent a request to a running daemon that performed LDAP search operations and caching on behalf of the library. This cleaner architecture meant that there was now only a single process that would need to open LDAP connections, running as a single user, and that if multiple requests came in simultaneously for the same piece of data, it could be funnelled into a single request.

A similar model was devised for Arthur de Jong's nss-pam-ldapd project written in 2006. Arthur took the nss_ldap library that Luke Howard originally developed, cleaned up the code and split it into a thin library and a separate running daemon that would handle all LDAP operations.

These architectural improvements to the way that the NSS library worked were not fundamental, and continued to be designed around RFC 2307 and RFC 2307bis.

This is the part where I first come into the story. I had spent many years since the late 90s working with LDAP-based solutions, including using but not really questioning the RFC 2307 schema. Then I stumbled upon a large NIS migration project in a bank. It was around about 2010, and this bank was merging two large NIS domains into a single domain, and migrating to LDAP. The problem with the domain merge was that were a lot of clashing UIDs and GIDs. This meant that a user in one part of the new organisation, say he has the user name *sam* with a UID of 500, could not login to a computer in the other part of the organisation as user *david* in the other domain already had a UID of 500. The solution was not to be found in RFC 2307, nor in any Open Source software, but in a commercial product that allowed UIDs to be remapped depending on which computer you logged into. As the storage was never going to be shared between computers in the different parts of the organisation, this worked fine. Sam

could have processes and files owned by him (UID 500) in his home group, but when he logged into a computer in the other group, although he had the same login credentials his processes and files would bear a different UID, say 4500. David had a similar remapping that occurred in the other direction. The solution was quite elegant and fitted the use-case well, but was non-standard, and as a result although a number of products offered this capability, they did so in incompatible ways. This lack of standardization led to vendor lock-in.

It was also at this time that I noted the duplicate configuration required to get a Linux client to use LDAP for name services as well as the automounter, and I witnessed for the first time a domain with over 10,000 netgroup entries. Netgroups were designed in NIS to group together users and hosts, and they can also inherit (include) from other netgroups. However, with over 10,000 entries in a flat namespace, it was naming convention alone that was being used to determine which netgroups applied where, and it was an extremely difficult task to determine which was the correct netgroup one should be added to to gain access to a particular resource, and to audit who had access to what.

However, I did nothing to address these issues at the time, but I parked them in the back of my mind where they would slowly stew. Now wind forward a few years to 2013, where I came across another NIS-to-LDAP migration project in a different bank. Here there was a new problem. In a number of maps there were identical entries, except one was in lowercase and one was in mixed or uppercase, and they were distinct entries. Examples could be found in the *passwd*, *group*, *netgroup* and *services* maps. So, for example, a service called *foo* on TCP port 400 is distinct to a service called FOO on TCP port 800. This works fine in NIS, because NIS, like UNIX file names, is case sensitive.

Unfortunately, RFC 2307 made generous use of the LDAP *cn* attribute, short for *commonName* and defined as case insensitive. This would prevent two entries *foo* and *FOO* in the same NIS map from being successfully imported into LDAP. This problem had been encountered by others, but had not been solved by a standard approach. Approaches included changing the data, patching the LDAP software, modifying the *cn* definition in the schema to be case sensitive and defining a new set of attributes and object classes that could be used in place of *cn*. The first approach was not possible as tracking down and modifying all uses of these entries in a large enterprise is very time-consuming and far from an exact science, the second approach would lead to a fork of the directory server software that would be costly to maintain, the third would result in a directory server that could only be used for a limited set of use-cases, so the fourth is the approach we took.

To achieve this, we defined three new custom attributes and seven new custom object classes, each with their own unique OIDs. We had had to deviate from RFC 2307 in order to represent data that was perfectly permissible within NIS. The RFC 2307 and NIS data schemas were therefore not 100% cross-compatible. Other people, I thought, must have had the same problem, and had to solve it in a similar yet incompatible way.

This time, I decided, a better solution was needed. I had encountered multiple issues with RFC 2307 and RFC 2307bis, so now was the time to do something about it. I would no longer sit on my hands and moan, I would put them on my keyboard and fix the problem once *and for all*.

Approach

So then, given the problem set, what was the best approach? RFC 2307 had deficiencies within it, which RFC 2307bis had carried through and not attempted to address. Some of these problems were in the documents themselves, while some were in the way that vendors had opted to implement the protocols.

I took a long hard look at the documents in question, and decided that whatever I did it would entail a lot of heavy shifting, because RFC 2307 was long and covered a lot of subject areas within a single document. In a similar vein to re-architecting the original nss_ldap library by splitting it into a thin client library and a server process, it would be easier to work with I thought, if RFC 2307 were split into a series of related documents each able to discuss the implementation of a set of related maps. Then there would need to be a document provided up-front that described the framework, how the other documents related to each other, as well as making it extensible such that new map types could be added easily in the future.

DBIS (pronounced *dee-biss*) was born. It began life as RFC 2307 (and the best bits from RFC 2307bis) split into separate documents. I could have called this RFC 2307ter, I suppose, but that didn't sound so great to me, and I needed a name that could not only be used to refer to the schema, but also to a new architecture model, and ultimately to Open Source software that would implement the new protocols. I opted for DBIS, which stands for Directory-Based Information Services, because of its similarity in spelling to NIS, for which it was intended to supersede and yet remain compatible with.

DBIS was to solve all of the problems identified in my background introduction. I set myself a number of rules that I must follow:

- I must be able to migrate all of my NIS data without having to change it (including in regard to case sensitivity).
- I must be able to use a DBIS client alongside a traditional nss_ldap library to provide one migration option.
- I must be able to use an RFC 2307 schema and an RFC 2307bis schema from a DBIS client.
- I must be able to use the basic DBIS schema from an RFC 2307 client, however, it is permitted to have extra features not available unless you use the DBIS client.
- I must not redefine any attributes or object classes that are already defined in RFC 2307 or RFC 2307bis. If I find a definition that is not right, I must invent a new one to replace it, rather than attempting to modify an existing one.

After writing a new set of internet drafts in August 2013, I started work on an Open Source reference implementation, found at http://dbis.sf.net. This consists of a daemon called dbis-cachemgr, written in Python, that is responsible for handling all LDAP communication on behalf of nss_dbis, the DBIS command-line client tool, and the Python, Perl and C APIs. The NSS library, nss_dbis, is the thin NSS library that Arthur de Jong split off from the original nss_ldap code and taken from the nss-pam-ldapd project. It is compiled to talk to the dbis-cachemgr daemon.

The reference implementation was finished in 2015, and has been improved upon slowly since its release. The latest addition is the C API, to go alongside the Python and Perl APIs, allowing applications such as Linux autofs and sudo to make direct use of dbis-cachemgr to gain access to information that is not otherwise available via the standard C NSS functions.

Configuration maps

I decided early on that there must be a standard way of pulling in data from multiple places within a DIT in order to produce one coherent map. I called this the *configuration map*, this would provide a natural hub through which configuration options could be applied, such as rules for remapping the names of attributes and object classes, rules for transforming UIDs etc., and rules for identifying which hosts get a particular view of a map.

In fact, further to this, I decided I should be able to mix schemas, not just between maps, but

within a map. I could have some entries populated within say the *passwd* database that originated from entries defined using RFC 2307, while having other entries defined using the DBIS (or some other) schema. This flexibility increases the number of options available to migrate from one to the other.

The type of configuration map entry is identified by its object class. So, for example, an entry with the object class dbisPasswdConfig will add one or more DNs to the list of where to get *passwd* entries from. Multiple dbisPasswdConfig entries are permissible, each with its own set of options.

All configuration map object classes have a super-class of dbisMapConfig, which is defined in *draft-bannister-dbis-mapping*. There are a number of generic attributes that can be applied to any configuration map entry, which enable various features, and these too are defined in the same document. The dbisPasswdConfig object class, and any features that are specific to *passwd* maps, are defined in *draft-bannister-dbis-passwd*, for example dbisMapGecos which defines the attribute name to use to get the value of the "gecos" field in the *passwd* database.

Case-sensitive attributes

The problem of case insensitivity has been solved in DBIS by defining a set of replacement attribute types. These are summarised in the table below. This is not a full list of new attributes introduced by the DBIS schema, only those introduced to resolve the case sensitivity issues.

Original Attribute	*DBIS Attribute*
cn (commonName)	en (exactName)
memberUid	exactUser
nisNetgroupTriple	netgroupTriple
memberNisNetgroup	exactNetgroup
ipServiceProtocol	ipProtocolName
nisMapName	customMapName

Replacement object classes

Modifying an object class already written down in an RFC is not permitted, therefore a set of replacement object classes were required, which are summarised in the table below. This is not a full list of new object classes introduced by the DBIS schema, only those introduced to replace existing object classes.

Original Object Class	*DBIS Object Class*
posixAccount	posixUserAccount
posixGroup	posixGroupAccount
nisNetgroup	netgroupObject
ipService	ipServiceObject
ipProtocol	ipProtocolObject
oncRpc	rpcObject
ipHost	ipHostObject
ipNetwork	ipNetworkObject
automountMap	automountMapObject
automount	automountEntry
nisObject	customMapEntry

The full list of new attributes and object classes and the rationale for their introduction can be found at http://sourceforge.net/p/dbis/wiki/DBIS%20and%20RFC2307%20schemas/.

Transformation rules

DBIS allows data to be transformed dynamically, client-side, according to business need. It currently supports four different types of transformation: text prefix, text suffix, numerical increment, numerical decrement; that can be applied to one or more attributes processed by any configuration map. Rules are added with the dbisTransAttr attribute. More than one transformation rule may be applied to a single attribute.

Overlays

Entire attributes may be replaced with alternative values for a set of users and groups by using the overlay feature. Coupled with netgroup constraints, this provides a way of varying fields in the *passwd* and *group* database depending on which host in the enterprise you login to.

Netgroup constraints

Another powerful feature of DBIS is the ability to restrict a configuration map to a list of hosts in a given set of netgroups, or the logical inverse of the set. This can be achieved using the exactNetgroup or notNetgroup attributes applied to a dbisMapConfig entry for any database, and in effect allows you to build different "views" of a map.

This introduces a number of possibilities that could not be achieved with an RFC 2307 configuration. For example, you can restrict a set of user and group accounts so that they only appear in the *passwd* and *group* entries on a subset of hosts, but share the same login domain as the rest of the enterprise. You can rewrite some UIDs and GIDs with transformation rules or overlays on only the set of hosts where ID clashes are known to exist. You can have application-specific TCP service name and port numbers registered only on the set of hosts that run the application that needs them.

Constraints are expressed by netgroup membership, so that to add that view or transformation to a particular host requires only for it to be added to the appropriate netgroup.

Netservices

To reduce the complexity of an environment with many thousands of netgroups, DBIS introduces a complementary structure called *netservices*. It works similarly to netgroups, except that it has a slash-separated hierarchical naming scheme and is intended to model application roles, privileges and services. Netgroups are intended to continue to group together collections of users and hosts, and are assigned to netservices.

For example, in DBIS you could define a netservice called ssh:login and assign netgroups of hosts identifying which hosts are allowed to run sshd and netgroups of users identifying which users are allowed to login. You could define a netservice called sudo:apache/httpd and sudo:root/ALL, and assign respective netgroups identifying who can use the sudo program to launch the httpd process as the user apache, and who can use sudo to run any command as the root user. You could define a netservice called ntp:ntpd and dns:bind and assign netgroups of hosts to identify to a configuration management system that those hosts should be running an NTP server and a DNS server respectively.

Membership of a netservice is queried using a new API innetsv() which works very similarly to innetgr(). Netservices are enumerated with getnetsvent() which is equivalent to getnetgrent().

Other miscellaneous schema improvements

A *passwd* entry no longer has a gecos attribute that must be independently configured, instead an existing attribute may be used for setting the gecos field and the name of that attribute is provided via the dbisMapGecos attribute assigned to the configuration map entry.

The automounter schema introduced by RFC 2307bis has been overhauled to make better use of different attributes for the various fields, to support multiple mount entries, different automounter map types, and including automounter maps within another.

DBIS netgroup entries may now be expressed as separate user and host elements instead of triples, making it easier to construct LDAP search expressions to find the netgroups you need. The third component of the netgroup triple has been slightly redefined in DBIS to represent the domain in which the user or host resides.

New attributes and abstract classes have been defined to differentiate between IPv4 and IPv6 addresses in host entries.

LDAP alias objects are now supported for expressing aliased entries, instead of multi-valued attributes.

Entries in any map may be disabled by setting the boolean attribute disableObject.

NIS-style custom map entries do not need to repeat the map name for every entry, as the name of the custom map is defined by the dbisMapName attribute assigned to the dbisMapConfig object.

DBIS Reference Implementation

The reference implementation available at http://dbis.sf.net is a fully functional software product that compiles and installs on various platforms including Linux and Solaris. It is included with a comprehensive test suite that executes over 1,000 discrete tests by mocking the LDAP library.

The latest version at the time of writing is DBIS 1.5.0, released at the end of September 2015, with more enhancements planned in forthcoming versions. The product consists of the components listed in the table below.

Component	*Description*
`dbis-cachemgr`	Multi-threaded daemon process that listens for client requests on a couple of UNIX domain sockets, handles all LDAP communication and caches results.
`nss_dbis`	Lightweight NSS library that converts NSS lookup requests to commands for dbis-cachemgr, and sends back the results.
`dbis`	Client tool that extends the functionality of the classic `getent` program to query all maps supported by DBIS and in a variety of output formats, including short & long text variants and JSON.
Python API Perl API C API	APIs in a variety of languages that provide a programmatic interface for querying data from `dbis-cachemgr` instead of going through the NSS library. This gives access to the extra maps supported by DBIS, as well as the extra variety of output formats.
Pyloom	Python library invented for DBIS that can be used by any Python program to simplify developing a multi-threaded server application. This library was developed because the existing Python libraries available for setting up multi-threaded TCP servers were lacking the scalability and flexibility required by DBIS.

Future improvements to DBIS are planned, in which YOU could help, including:

- Autofs integration, so that the standard Linux automounter can make use of the DBIS C APIs to obtain map data.
- Sudo integration, so that sudo can make use of Netservices as well as Netgroups to add greater flexibility and visibility into the roles that people have.
- Puppet integration, so that the Netservices assigned to groups of hosts can determine how a system is configured.
- Expanding the definition of custom maps to better represent multi-column custom data.
- Providing some standard packages for various platforms and releases.
- Add support for LDAP persistent searches so that certain maps could be updated immediately and automatically by `dbis-cachemgr` whenever they are changed with callback mechanisms for interested client software.
- Migration tools.
- Add more language support, e.g. Java API.
- Add more load-balancing algorithms.
- Add support for more LDAP authentication schemes.
- Allow different LDAP server profiles to be defined that can be used by different maps. This would allow some data to originate from different directories or with different

credentials and load-balancing algorithms.

- Ultimately `dbis-cachemgr` could become the defacto standard way that applications can retrieve reference data over LDAP, reducing the complexity of third-party software.

Further information

The IETF internet drafts relating to DBIS are as follows:

Name	*URL*
DBIS Mapping Objects	http://www.ietf.org/id/draft-bannister-dbis-mapping.txt
DBIS Netgroups and Netservices	http://www.ietf.org/id/draft-bannister-dbis-netgroup.txt
DBIS Users and Groups	http://www.ietf.org/id/draft-bannister-dbis-passwd.txt
DBIS Hosts, Networks and Services	http://www.ietf.org/id/draft-bannister-dbis-hosts.txt
DBIS Devices	http://www.ietf.org/id/draft-bannister-dbis-devices.txt
DBIS Automounter	http://www.ietf.org/id/draft-bannister-dbis-automounter.txt
DBIS Custom Maps	http://www.ietf.org/id/draft-bannister-dbis-custom.txt

The DBIS wiki contains a lot of information: http://dbis.sf.net

My blog is also useful: http://technicalprose.blogspot.co.uk/2013/08/introducing-dbis.html

Find me on LinkedIn: https://uk.linkedin.com/in/mbannister

DBIS has also been discussed at intervals on the IETF ldapext mailing list: http://www.ietf.org/mail-archive/web/ldapext/current/maillist.html