VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

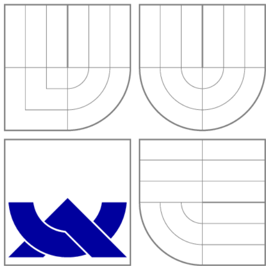
IMPLEMENTATION OF NIS BACKEND FOR SSSD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

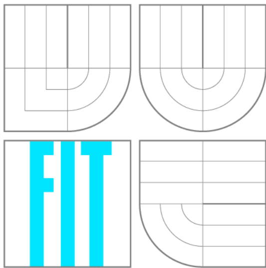
Bc. LUKÁŠ NYKRÝN

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE NIS BACKENDU DO SSSD

IMPLEMENTATION OF NIS BACKEND FOR SSSD

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. LUKÁŠ NYKRÝN

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN ZELENÝ

SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se v první části zabývá představením technologií a nástrojů pro centrální správu a přihlašování uživatelů v GNU/Linux. Ukazuje využití adresářových služeb v síťové infrastruktuře, konkrétně služby NIS a její porovnání s dnes pravděpodobně nejrozšířenější adresářovou službou LDAP. Dále práce popisuje proces autentizace na klientských stanicích, konkrétně použití PAM a NSS a možné rozšíření celého systému zavedením cache díky démonu SSSD. Druhá část popisuje návrh a implementaci NIS provideru pro SSSD.

Abstract

The first part this thesis introduces technologies and tools for centralized management and authentication of users in GNU / Linux. It shows the usage of directory services in a network infrastructure, namely the NIS and its comparison with today probably the most widely used directory service LDAP. Then it describes the process of authentication on client workstations, specifically use of PAM and NSS, and possible expansion of whole system through the introduction of cache by using daemon SSSD. The second part of this thesis describes design and implementation of the NIS provider for SSSD.

Klíčová slova

bezpečnost, NIS, LDAP, SSSD, PAM, NSS, GNU/Linux, síťová autentizace,
správa uživatelů

Keywords

security, NIS, LDAP, SSSD, PAM, NSS, GNU/Linux, network authentication,
user management

Citace

Lukáš Nykrýn: Implementation of NIS Backend for SSSD, diplomová práce, Brno, FIT VUT v Brně, 2013

Implementation of NIS Backend for SSSD

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Zeleného

. . . .
Lukáš Nykrýn
May 20, 2013

Poděkování

Děkuji svému vedoucímu Ing. Janu Zelenému za veškerou pomoc a rady, které mi poskytl během vypracování této práce.

© Lukáš Nykrýn, 2013.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Contents

1 Introduction **4**

2 Directory services **5**
- 2.1 NIS . . . 5
  - 2.1.1 Structure . . . 5
  - 2.1.2 Communication and daemons . . . 6
  - 2.1.3 Maps . . . 7
  - 2.1.4 NIS setup . . . 10
  - 2.1.5 NIS+ . . . 12
- 2.2 LDAP . . . 13
  - 2.2.1 Information model . . . 13
  - 2.2.2 Naming model . . . 14
  - 2.2.3 Function model . . . 15
  - 2.2.4 Security model . . . 16
- 2.3 Comparing, use cases and transferability . . . 17

3 GNU/Linux user management **18**
- 3.1 PAM . . . 19
  - 3.1.1 Configuration . . . 19
- 3.2 NSS . . . 20

4 SSSD **23**
- 4.1 SSSD Features . . . 23
- 4.2 Configuration . . . 24
- 4.3 Architecture . . . 24
  - 4.3.1 Design of provider . . . 25

5 Design **27**
- 5.1 Features to implement . . . 27
  - 5.1.1 Users . . . 27
  - 5.1.2 Groups . . . 28
  - 5.1.3 Services . . . 28
  - 5.1.4 Netgroups . . . 28
- 5.2 Communication with NIS . . . 29
  - 5.2.1 RPC communication with NIS server . . . 29
  - 5.2.2 Communication through `ypbind` . . . 30
- 5.3 Handling of requests . . . 31

6 Implementation 33
6.1 Initialization 33
6.2 Authentication requests 34
6.3 ID requests 37
6.3.1 User 37
6.3.2 Group 37
6.3.3 Service 38
6.3.4 Netgroup 38
6.4 Communication with NIS 40
6.4.1 Preparation of the request 40
6.4.2 Spawning child 42
6.5 Child process and queries to NIS 43
6.5.1 Initialization of child 43
6.5.2 Query to NIS database 43
6.5.3 Return of values from child 44

7 Testing 46
7.0.4 ID provider 46
7.0.5 Authentication provider 46

8 Conclusion 48

List of Figures

2.1 Scheme of NIS domain 6
2.2 Elements of information model 14
2.3 Naming model 15

3.1 Accessing data before introduction of PAM and NSS 18
3.2 Usage of PAM and NSS 22

4.1 Scheme of SSSD architecture 24

5.1 Communication in the provider 32

6.1 Authentication request 36
6.2 ID request 39
6.3 Key buffer 41
6.4 Enum buffer 41
6.5 Buffer with reply to lookout request 45
6.6 Buffer with reply to enum request 45

Chapter 1

Introduction

Computers in these days became one of the most important tools in our lives and for example in a company environment, they store huge amount of classified information, so we need to make sure, that every computer user is authenticated.

Unlike the past where computer networks were organized in the server-terminal model and the server did the whole authentication process, today in the server-client model part of this process is done by client computers. This brings the question how can a client computer determine, if username and password provided by user are correct and what privileges should the user have.

Because today's networks contain hundreds of users and client computers, it is impossible to create database of all users on every computer manually, so we need to define a storage which will contain all of the information and define, how client computers can perform queries against this storage.

The extensive usage of mobile devices, especially laptops, brings an another problem, how can be the user authenticated, when his computer is temporarily not connected to a company network.

The first part of this work should provide a description of NIS and a comparison with today's most used directory service LDAP.

The second part describes the current state of GNU/Linux account management, concretely PAM and NSS.

The third part presents SSSD and it shows benefits from its introduction to the system.

The fourth part discusses ways, how can be the NIS provider for SSSD implemented.

The fifth part contains a description how was the provider implemented.

The sixth part describes methods to test the provider.

The conclusion summarizes the created solution, it discusses its pros and cons and shows possible improvements.

Chapter 2

Directory services

Directory services provide an access to information stored in databases called *directories*. The word directory in this context has nothing to do with directories in a file system, it is a specific type of database, which stores various information organized in groups, for example telephone numbers, user logins, computer names, ...

In the contrast to relational databases, these are designed for a frequent reading and searching and only for an occasional modification and also data in these can be stored redundantly, if it helps with a performance.

There are many applications, which can be classified as a directory service for example LDAP, NIS, X.500, DNS, Hesiod and NetInfo.

This chapter will be dedicated to the two most common directory services, which can be used to authenticate users: LDAP and NIS.

2.1 NIS

NIS (Network Information Protocol) is a distributed directory service protocol, which is designed to share various system configurations in a network. It was designed by Sun Microsystems in 80's as YP (Yellow pages) but it was changed because this name is registered trademark of British Telecoms. But this is the reason, why most of NIS tools begin with a prefix "yp".

2.1.1 Structure

All computers which are sharing information through NIS belong to the same *domain* and this name is completely independent on DNS. In every domain there are three types of machines.

- **Master server** is a single machine in every NIS domain, that holds and creates authoritative copy of NIS maps (see chapter 1.1.3) and propagates changes to slave servers.
- **Slave servers** are additional servers. They receive complete read-only copies of maps from the master and offer the same services like the master server for clients. Their main purpose is to create backup for master server and clients can connect to them, if other server is unavailable due to a high load.

- **Clients** are machines, that request information from maps on servers. They does not need to know, if the server, which they are asking, is master or slave since both of them have complete copies of all maps.[15]

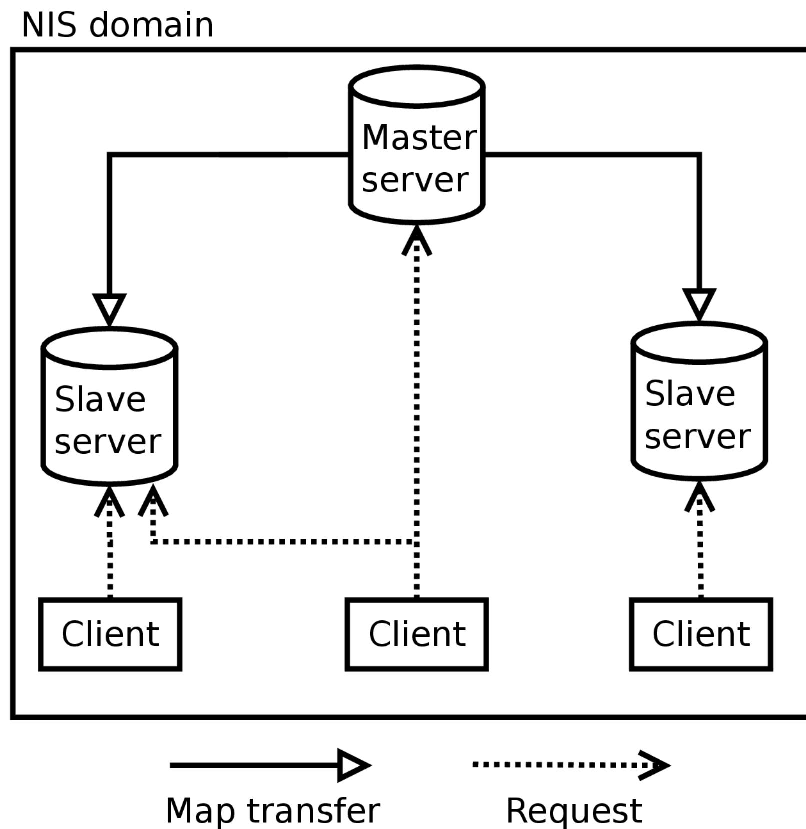


Figure 2.1: Scheme of NIS domain

2.1.2 Communication and daemons

The communication between machines in NIS domain is completely based on RPC. There are four daemons in NIS environment: `ypbind` running on client, `ypserv` running on all servers and `ypupdated` and `yppasswdd` running only on master server.

`ypbind`

Every system that runs `ypbind` is automatically a NIS client. Main purpose of `ypbind` is to bind to the NIS domain. After it starts, it finds, through broadcast message, a server which is responsible for supplying information for a client's domain and periodically checks, if this that server is still running and his responses are not slow due to a high load. If this server is unavailable, `ypbind` sends the broadcast message again to find another one. Other client's processes can connect to the NIS server through `ypbind`.

`ypserv`

All of NIS services are provided by daemon `ypserv`. These we can splits to three types: data lookups, map maintenance calls, and NIS internal calls. Since information in NIS maps are stored in the key→value format, lookup requests are also key oriented.

We have four queries, which can be performed:

- **Match** finds the corresponding value to a key.
- **First** returns the first pair of key-value in a map.
- **Next** returns the next pair of key-value in a map.
- **All** is used to obtain the whole map from a server.

The maintenance calls are used to obtain information that are needed to perform map transfers from a master to slave servers.

- **Order** returns the creation date of a particular map.
- **Master** obtains the address of master server for this map.

`Ypserv` is not aware for which domains and maps it is providing information, it simply looks in `/var/yp/$DOMAIN` and there obtains information from the desired map. This brings possibility to host information for multiple domains on one server.

`yppasswdd`

`Yppasswdd` daemon brings possibility to change user's password, full name or shell. It accepts an incoming request, authenticates it and updates password maps.

`ypupdated`

The daemon `ypupdated` prompts slave servers to update their copies of maps. Which maps should be updated is determined from the *updaters* file.

2.1.3 Maps

NIS maps are catalogs of all information, that NIS provides and they are physically stored in DBM databases. These maps are mostly generated from regular configuration files with `makedbm` in "key → value" format. In this type of storage it is possible to find information only through a key, so the same configurations are usually stored in multiple maps with a different key, for example user's passwords are stored in `passwd.byname` and `passwd.byuid`.[13]

DBM (DataBase Manager) databases are one of the first databases engines, developed in 1979 by AT&T. DBM database consists of a set of keys and associated values, organized in a hash table. Their design allows to obtain the required value in just two accesses to a filesystem, which brings great performance improvement against reading a normal configuration file.

Each DBM database, and therefore each NIS map, comprises two files: a hash-table accessed bitmap of indexes and a data file. The index file has the `.dir` extension and the data file uses `.pag`.[16]

Map	Nickname	Source file	Description
passwd.byname	passwd	/etc/passwd	Contains password information with user name as key.
passwd.byuid			Same as passwd.byname, except that key is user ID.
group.byname	group	/etc/group	Contains group security information with group name as key.
group.bygid			Contains group security information with group ID as key.
hosts.byaddr	hosts	/etc/hosts	Contains machine name, and IP address, with IP address as key.
hosts.byname			Contains machine name and IP address, with machine (host) name as key.
ethers.byaddr	ether	/etc/ethers	Contains machine names and Ethernet addresses. The Ethernet address is the key in the map.
ethers.byname			Same as ethers.byaddr, except the key is machine name instead of the Ethernet address.
networks.byaddr	networks	/etc/networks	Contains names of networks known to your system and their IP addresses, with the address as key.
networks.byname			Same as networks.byaddr, except key is name of network.
rpc.bynumber		/etc/rpc	Contains program number and name of RPCs known to your system. Key is RPC program number.
services.byname	service	/etc/service	Lists Internet services known to your network. Key is port or protocol.

Table 2.1: NIS default maps [14]

Map	Nickname	Source file	Description
protocols.byname	protocols	/etc/protocols	Contains network protocols known to your network.
protocols.bynumber			Same as protocols.byname, except that key is protocol number.
netgroup.byhost		/etc/netgroups	Contains group name, user name and machine name.
netgroup.byuser			Same as netgroup.byhost, except that key is user name.
netgroup			Same as netgroup.byhost, except that key is group name.
bootparams		/etc/bootparams	Contains path names of files clients need during boot: root, swap, possibly others.
mail.aliases	aliases	/etc/aliases	Contains aliases and mail addresses, with aliases as key.
mail.byaddr			Contains mail address and alias, with mail address as key.
netid.byname		/etc/passwd, /etc/groups, /etc/hosts, /etc/netid	Used for UNIX-style authentication. Contains machine name and mail address (including domain name). If there is a netid file available it is consulted in addition to the data available through the other files.
netmasks.byaddr		/etc/netmasks	Contains network mask to be used with IP submitting, with the address as key.
ypservers			Lists NIS servers known to your network.

Table 2.2: NIS default maps (continue) [14]

2.1.4 NIS setup

Setup of NIS is quite simple. Preparation of master server, slave server and clients can be done in few steps.

Master server

1. **Setting the NIS Domain Name**
   This can be done temporarily through command:

   ```
   domainname name_of_domain
   ```

   or by modifying `/etc/sysconfig/network` to:

   ```
   NIS_DOMAIN=name_of_domain
   ```

2. **Starting `ypserv` daemon**
   Next step is to run `ypserv` daemon, which is responsible for handling NIS queries:

   ```
   /etc/init.d/ypserv start
   ```

3. **Altering makefile for creating maps**
   Transferring configuration files to NIS maps is done by running `make` on `/var/yp/Makefile`. Most important setting are:

   - Variable ALL sets, which configuration files will be transfered:

     ```
     ALL = passwd group hosts rpc services netid protocols netgrp
     ```

   - Location of configuration files:

     ```
     YPPWDDIR = /etc

     GROUP       = $(YPPWDDIR)/group
     PASSWD      = $(YPPWDDIR)/passwd
     SHADOW      = $(YPPWDDIR)/shadow
     GSHADOW     = $(YPPWDDIR)/gshadow
     ALIASES     = /etc/mail/aliases
     ```

   - Allowing pushing maps to slave servers:

     ```
     NOPUSH=false
     ```

4. **Running `ypinit`**
   After altering `/var/yp/Makefile` we can initialize NIS server by running:

   ```
   /usr/lib/yp/ypinit -m
   ```

   where parameter `-m` says, that this will be the master server.

Slave server

1. **Setting the NIS Domain Name**
   This can be done temporarily through command:

   ```
   domainname name_of_domain
   ```

   or by modifying `/etc/sysconfig/network` to:

   ```
   NIS_DOMAIN=name_of_domain
   ```

2. **Adding address to the master server**
   To ensure that maps will be propagated to this slave server, we must add his address to `/var/yp/ypservers` on master server.

3. **Starting `ypserv` daemon**
   Next step is to run `ypserv` daemon, which is responsible for handling NIS queries:

   ```
   /etc/init.d/ypserv start
   ```

4. **Running `ypinit`**
   Now we can initialize slave server by running:

   ```
   /usr/lib/yp/ypinit -s master_server
   ```

   where parameter `-s`, says that this will be the slave server and `master_server` is the address of the master server.

Client

1. **Setting the NIS Domain Name and server**
   In file `/etc/yp.conf` we must specify name of domain and how `ypbind` determines address of a server.

   ```
   domain name_of_domain server server_address
   ```

   or

   ```
   domain name_of_domain broadcast
   ```

2. **Starting `ypbind` daemon**
   Final step is to run `ypbind` daemon:

   ```
   /etc/init.d/ypbind start
   ```

[9]

2.1.5 NIS+

In 1992 Sun Microsystems introduced NIS+, which was designed as NIS successor. It brings features that were missed inthe original NIS, but it is more difficult to administer on server side and in GNU/Linux it has buggy client and no usable server.

Hierarchy

In NIS you can split informations to multiple domains, but these domains are flat and it is not natively possible to share informations between them. NIS+ can behave similarly as DNS and you can order domains to hierarchical structures.

Security

Biggest disadvantages of NIS is lack of security. All information are available to everyone, transmissions are not encrypted and there is no authentication between client and server. NIS+ is build above RPC/DH, so it solves problems with with encryption of transmission and authentication.

Updating

Generally in NIS every update means that administrator alters configuration files, transforms them to DBM files and database is pushed to slave servers. NIS+ offers possibility to change informations by user and the ability to propagate these changes by incrementation updates.[13]

2.2 LDAP

LDAP (Lightweight Directory Access Protocol) is a simplified version of X.500, which is collection of standards developed by International Consultative Committee of Telephony and Telegraphy. Whole system can be described from four views, which are in LDAP terminology called models.

- **Information model** describes LDAP as a data storage. It defines, what data types can we store, mechanism how are these data stored and operation which can be performed over them. This information for a specific directory creates the *directory schema*.
- **Naming model** defines hierarchical structure over data. In LDAP every record is identified by *Distinguished Name (DN)*, which can be defined as a path from root of LDAP domain to the record.
- **Functional model** deals with access to data. It describes operation, which can be performed over LDAP.
- **Security model** describes what needs to be done to get an access to data in LDAP.

2.2.1 Information model

Information model describes basic units of LDAP called *entries*. Every entry is composed from attributes, which are basically properties of an object, which is represented by entry.

Every entry can contain multiple *attributes* and every attribute has a *type* and its *value(s)*. For every type of attribute we have also definition of *operations*, that we can perform over these attributes. Attributes can be also distinguished as *user*, which describes properties of the object, and *operational*, which contains system information, for example time stamp.

Set of attributes also creates *object class*, which describes some object from real world. Between objects is inheritance which goes from abstract class *top*. Every class has unique identification *OID*, unique name, its ancestors, type, and list of mandatory and optional attributes. Object classes can be abstract, structural or auxiliar.

All classes and attributes are defined in *schemes*. Standardized schema are often included in installation of server.

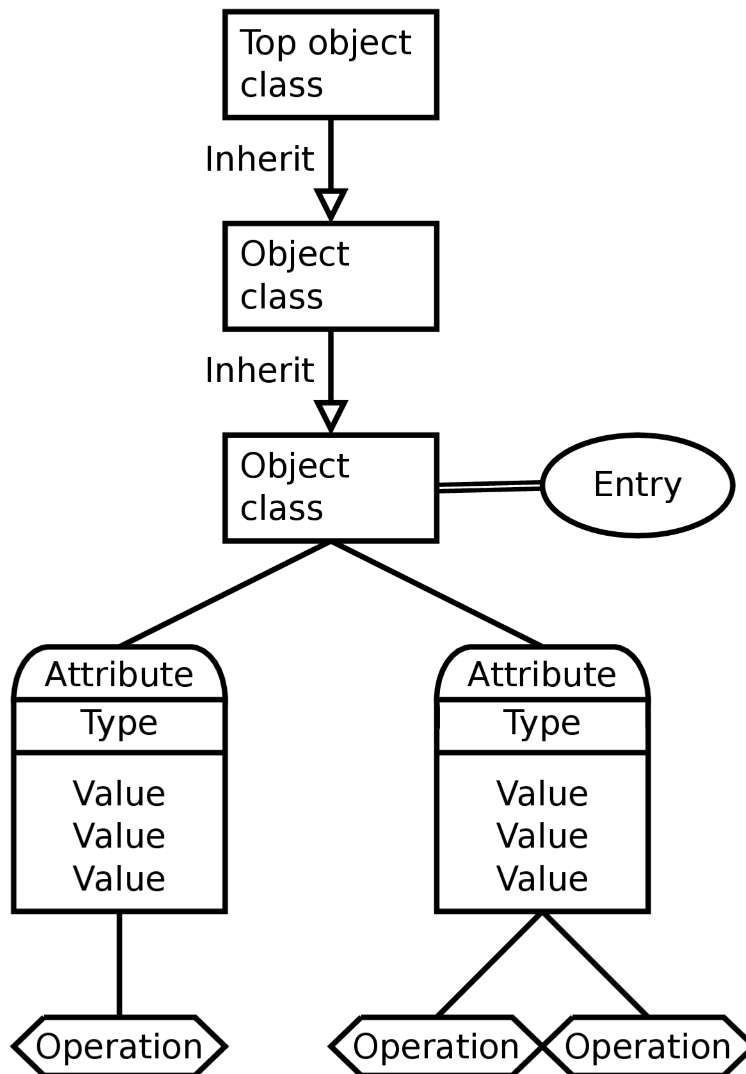


Figure 2.2: Elements of information model

2.2.2 Naming model

LDAP Naming Model defines, how data will by organized. This structure is called *DIT (Directory Information Tree)*. Basically, it is tree graph where leafs are entries representing entities in real world. Any node can be identified by *Distinguished Name (DN)*, which is defined by sequence of relative DN from root to node.

If a real structure can not be described by pure tree structure (for example in company is one person working in two departments), we can use *aliases*. These create link from leaf to another part of the tree.

Another type of link can be created by object *referral*. This shows URL to another LDAP server. If there is a query which must be performed over referral, it can be process by the server itself, so client does not know that information is stored elsewhere or server just gives client the URL to the other server and client send a query directly to this server.

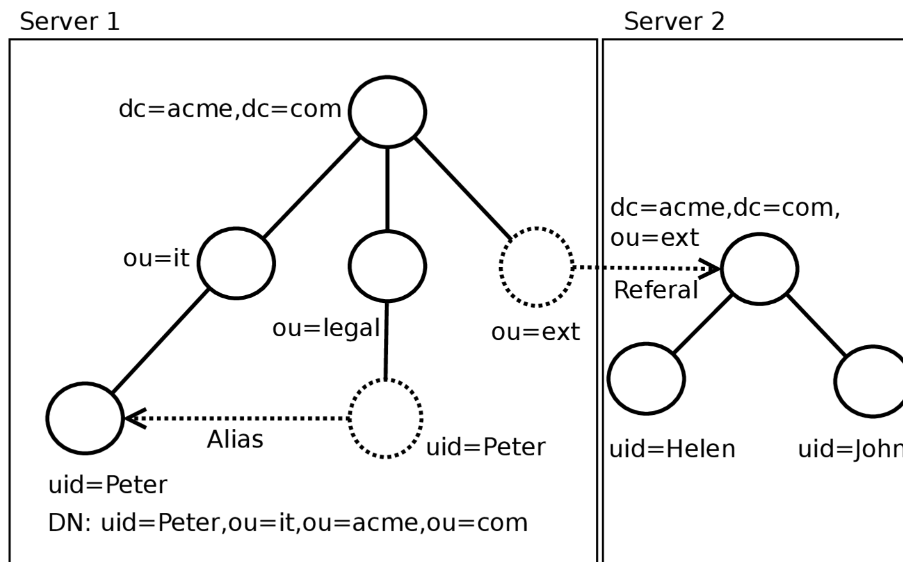


Figure 2.3: Naming model

2.2.3 Function model

Function model describes, which operations can be performed over LDAP: queries, data alteration and access control.

Queries

- **Search** is used to perform lookup in directory tree and returns whole entries or defined attributes which fit the search request. We need to specify 8 parameters:
  - **Base object** acts as root of a subtree, where the search will be performed.
  - **Scope** determines on which part of subtree, it will be performed. This can be only in the root object (*base*), in all descendants of root object (*onelevel*) or in whole subtree (*sub*).
  - **Alias dereferencing** sets what would be done, if the search encounters an alias. Options are *neverDerefAliases*, *derefInSearching*, *derefFindingBaseObject* and *derefAlways*.
  - **Size limit** sets maximum number of entries, which client will accept.
  - **Time limit** specify period in seconds, during which client is accepting the results.

- **Attributes-only** says if we want only list of attributes of entries, or if we want also their values.
- **Search filter** specifies what we want to find. Filters are *equality, substring, approximate, greater than, less than, presence* and bool operators *AND,OR,NOT*
- **List of attributes** determines, which attributes we want to return.

- **Compare** is used for testing values of attributes in an entry. According to match it return true or false. Parameters are DN of the entry and a list of attributes and their values.

Data altering operations

LDAP has four operations that can be use for modifying database.

- **Add** adds a new entry to database. It accepts DN of the new entry and lists of attributes and its values.
- **Delete** removes an entry from database.
- **Rename** change location of an entry in tree.
- **Modify** alters attributes in an entry.

Authentication and control operations

LDAP has two authentication operations and one control operation.

- **Bind** is used to attach to the server, sets negotiation method of authentication and identification of client. It provides possibility of plaintext authentication or through SASL with md5 hash.
- **Unbind** terminates connection to client.
- **Abandon** terminates previous operation.

2.2.4 Security model

Last model describes LDAP from a view of security. It describes methods of preventing unauthorized access to data stored in LDAP. According to type of authorization, we can divide all LDAP servers to three groups:

- **Anonymous authentication** – with this settings server does not ask for any credentials and offers data to everyone. This type of server should be read-only.
- **Password authentication** – this server must offer SASL authentication with MD5.
- **Encryption and authentication** – server must have support for TLS encryption and authentication through keys or certificates.

[8][1]

2.3 Comparing, use cases and transferability

Comparing

In most cases when somebody is building a new large computer network and there is need for directory service, LDAP is definitely better choice than NIS.

- **Security**
  As was mentioned above LDAP offers multiple types of authentification, access management and communication can be encrypted. NIS does not have any of these features. Access for modifiing data can be described as root or nothing, RPC communication is not encrypted and everybody has read access to all information stored in NIS maps.

- **Data structure**
  Ldap organizes data to tree structure and offers possibility to define items according to our needs. NIS maps are flat key-value tables with no possibility of hierarchical structure.

- **Performance** Comparison of performance of LDAP and NIS is not quite possible, since NIS provides only basic lookups while LDAP can perform more sofisticated queries. NIS can probably find value according to key faster, because this is primary purpose of its DBM databases, but when we need more complicated query, we must go throught whole map and process data on client, while LDAP can solve this itself.

Use cases

Despite this, there are to two cases where NIS is used in these days. First category are networks, which were built with NIS in the time when LDAP was not widely used, they have no extra security needs and migration can only brings a possibility of problems.

Second category would be new small networks, in which we just want to share for example usernames and passwords. In these LDAP can be unnecessarily robust and resource consuming.

Transferability

For a migration from NIS to LDAP there are a lot of tools which should automatize the process for example **MigrationTools** from PADL Software Pty Ltd.

For improbable migration from LDAP to NIS we can use LDAP tools for example *ldap_get_users* to generate configuration files (in this case *passwd*) and these can be directly used by NIS.

Chapter 3

GNU/Linux user management

In early years of UNIX systems all authentications were purely based on local files (`/etc/passwd, /etc/groups, ...`) and all programs which wanted to get information about users, or basically all system and network information, had to read these files and process them. This approach has many disadvantages. These files must be readable from whole systems (so for example everybody can retrieve hashes of user's password), every program must implement reading of these files, we can not simply use anything else then password authentication (fingerprints, certificates, ...) and all user's information must be in these files, so it is impossible to use another storage unless it is implemented directly in an application.

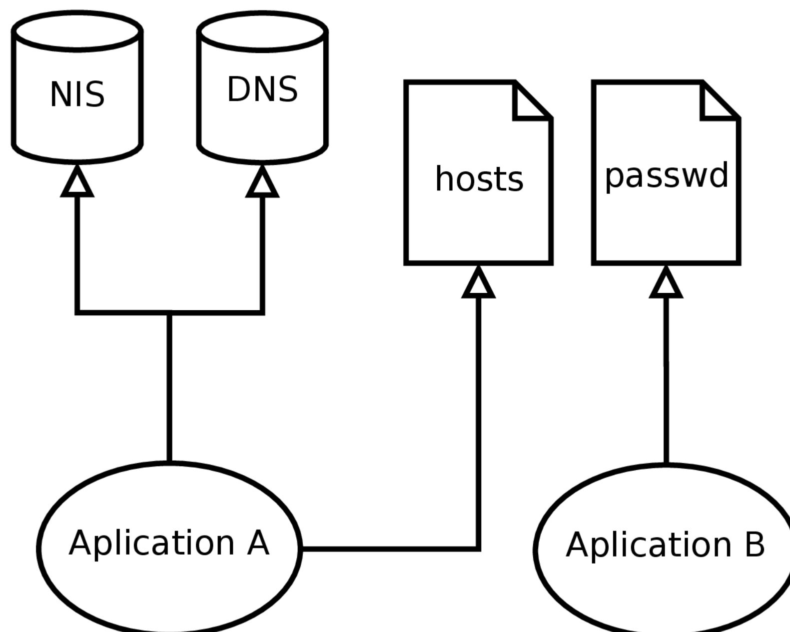


Figure 3.1: Accessing data before introduction of PAM and NSS

This can be solved by introducing an unified access to information about users. All modern Unix systems are typically using PAM for the user authentication and NSS for providing (among other things) information about him.

3.1 PAM

PAM (Pluggable Authentication Modules), developed by Sun Microsystems, allows to perform user authentication. PAM system consists two parts:

- Dynamically linked libraries which are usually located in `/lib/security`.
- A set of configuration files usually stored in `/etc/pam.d`.

Developer of application simply implements calling of several dynamically linked libraries, which are included in PAM and requests the authentication of a user and PAM performs all needed actions according to configuration file of this application.

Tasks, which are performed by PAM can be split into four groups:

- **Authentication** – PAM decides, if the user is, who he claims to be. This is typically done through password authentication, but other methods, like fingerprints, hardware tokens or certificates, are also possible.
- **Account management** – PAM provides methods to determine, if user is allowed to access some service. This area for example covers check of password expiration or determination, if user is member of a group which can use this service.
- **Password management** – PAM takes care about changing user's password or updating them if they expire.
- **Session management** – PAM also performs operations which stand besides the pure authentication process, for example mounting user's home directory or protocol events and maintaining the log files.

3.1.1 Configuration

As mentioned before, for every application, PAM looks to its configuration file, which is consisted by lines in format:
Module-type Control-flag Module-path Arguments

- *Module-type* – This defines type of the module according to to which part of PAM it belongs.
  - `auth`
  - `account`
  - `session`
  - `password`
- *Control-flag* – Since modules can be organized in groups, this defines what should be done in case of success or failure of this concrete module.
  - `required` – Failing in this modules results in complete failure and the next modules are proceeded.

- `requisite` – This has the same meaning as `required`, but process is stopped in case of failure.
- `sufficient` – Success in this module is enough to success in the complete process of authentication. If there are not any unprocessed `required` modules, the process is stopped.
- `optional` – This module is not required for complete success.

- *Module-path* – Describes path to the module, for example `/lib/security/pam_access.so`.
- *Arguments* – This optional value represents list of parameters, which will be pass on to module. There are some standardized parameters:
  - `debug` – This argument turns on writing debugging information to syslog.
  - `no_warn` – Suppresses all warning messages.
  - `use_first_pass` – Module does not ask user for password, but it uses password from previous module.
  - `try_first_pass` – This is similar as use_first_pass argument, but if authentication fails module asks for another password.

[10]

3.2 NSS

NSS (Name Service Switch) provides unified access to information stored in various databases.

Whole configuration of NSS is stored in `/etc/nsswitch.conf`, which describes, where NSS should look for concrete information. Information provided by NSS are from following areas:

- `aliases` – Mail aliases.
- `ethers` – Ethernet numbers.
- `group` – Groups of users
- `hosts` – Host names and numbers.
- `netgroup` – Network wide list of hosts and users, used for access rules.
- `networks` – Network names and numbers.
- `passwd` – User passwords.
- `protocols` – Network protocols.
- `publickey` – Public and secret keys for Secure_RPC.
- `rpc` – Remote procedure call names and numbers.
- `services` – Network services.
- `shadow` – Shadow user passwords.

Typical source databases are:

- `files` – File in `/etc` of client.
- `nis` – NIS map.
- `compat` – Service provides support for old +/- syntax.
- `dns` – Obtain host information from DNS.
- `ldap` – Information will be obtained from LDAP directory.

If there is specified only one source, NSS tries to find information there and according to the result it returns the corresponding message:

- `SUCCESS` – Information was correctly found.
- `UNAVAIL` – The source is not responding or is not available.
- `NOTFOUND` – Information was not found.
- `TRYAGAIN` – The source is currently busy.

If there are multiple sources, NSS tries to obtain information from the first source and if it is successful it returns `SUCCESS` message, in the other case it continues to the next source. If the information was not found in any source message `NONSUCCESS` is returned.

This behavior can be altered by adding action option. For example if we use

```
networks: nis [NOTFOUND=return] files
```

and the information is not found in a NIS map, the search is unsuccessful, but when the NIS is unavailable or busy, NSS will look to local configuration.[9]

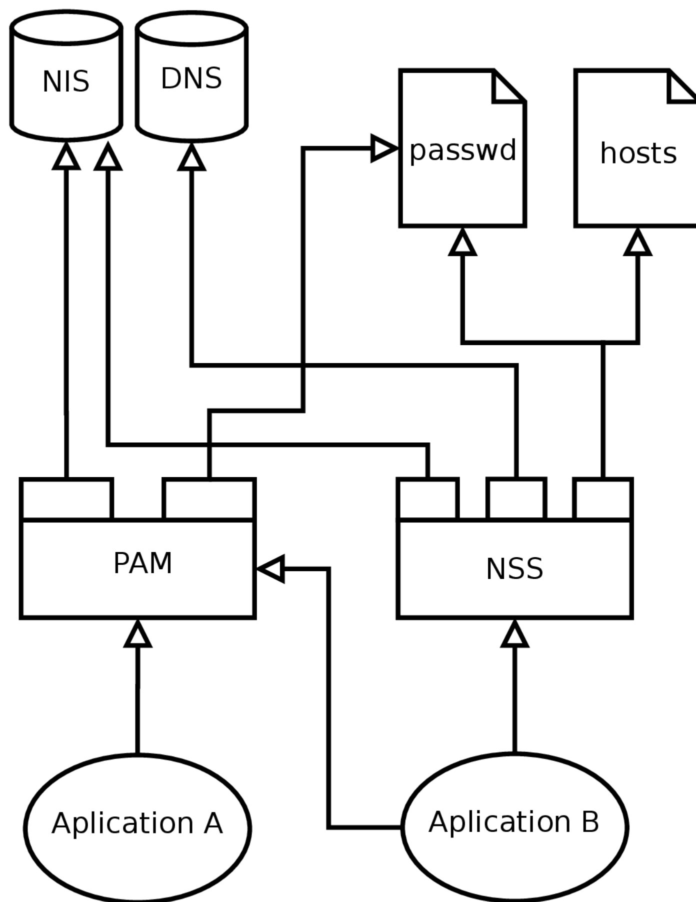


Figure 3.2: Usage of PAM and NSS

Chapter 4

SSSD

With mechanisms described in previous text, we can build centrally administered database of user's account and let the client computer to use it for authentication of users. But this design is not flawless.

Biggest disadvantage of network authentication is its dependence of network connection. When the computer is off-line, users can't authenticated themselves, so they are not allowed to access the system. This can be bypassed by creating local accounts on every computer, but in most scenarios this impossible due to difficult maintenance and security problems.

4.1 SSSD Features

The SSSD is a service which creates a "layer" between PAM/NSS and identity and authentication providers. It brings following improvements to the authentication architecture:

Off-line authentication

SSSD can solve the problem with dependence on permanent network connection by introducing cache.

- Only useful data are stored in cache.
- User can log in, even when he is outside the network or the server is down.
- For log attempt SSSD always tries to contact the server.
- Items in cache expire.

Support for multiple domains

SSSD can connect to multiple domains of the same type, this can be very useful for example with LDAP, where is setup of obtaining information from different domains complicated.

Decrease load of server

Another great asset of SSSD is help to reduce load on directory servers. For example when we have on client multiple applications, which are requesting data from LDAP, every one of them is opening its own connection and this can lead to overload of the server. SSSD can perform all these requests in single connection.

4.2 Configuration

Whole configuration of SSSD is stored usually in `/etc/sssd/sssd.conf`. This file consist from configuration of particular sections describing domains and services. Its format is similar to `.ini` file:

```
[section]
# Comment line
key1 = val1
key10 = val1,val2
```

[11]

4.3 Architecture

SSSD consists from several processes which are communicating through protocol DBus, internally called SBus.

- **Monitor** is a central process, which supervise other processes.
- **NSS responder** provides information about users which are requested from NSS module nss_sss.
- **PAM responder** performs communication between PAM module pam_nss and SSSD
- **Data providers** ensure communication with servers and they store data to cache. [12]

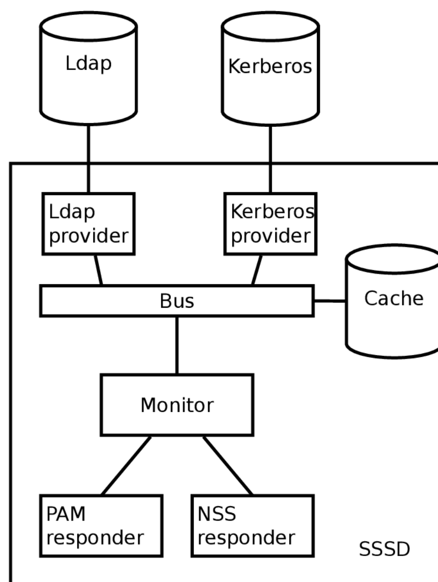


Figure 4.1: Scheme of SSSD architecture

4.3.1 Design of provider

Since the object of whole work is to design the NIS provider for SSSD, the architecture of provider deserves to be described in more details.

Every provider should implement four functions which are used for initialization:

```
int sssm_<provider_name>_id_init(struct be_ctx *bectx,
                                 struct bet_ops **ops,
                                 void **pvt_data);

int sssm_<provider_name>_auth_init(struct be_ctx *bectx,
                                   struct bet_ops **ops,
                                   void **pvt_data);

int sssm_<provider_name>_access_init(struct be_ctx *bectx,
                                     struct bet_ops **ops,
                                     void **pvt_data);

int sssm_<provider_name>_chpass_init(struct be_ctx *bectx,
                                     struct bet_ops **ops,
                                     void **pvt_data);
```

This functions can be compared to theirs equivalents in PAM and NSS.

- **ID** performs general NSS query.
- **AUTH** represents PAM authentication request.
- **ACCESS** handles PAM query about allowing access to various resources.
- **CHPASS** ensures PAM command to change user's password.

All important information for run of the provider (like connector to SBus, access to database or information about domain) are set in `struct be_ctx`.

Provider can set its "personal" data (for example connection to server) to `void **pvt_data`, these data can later be accessed through `be_ctx->bet_info[BET_ID].pvt_bet_data`.

Main goal of these functions is to populate the structure `struct bet_ops`, which has three members:

```
struct bet_ops {
    be_req_fn_t check_online;
    be_req_fn_t handler;
    be_req_fn_t finalize;
};
```

These pointers are for registration of callbacks, which will be called, when we need to check status of connection, perform a request or end the provider.

All of these function must accept one argument be_req, which consists following members:

```
struct be_req {
    struct be_client *becli;
    struct be_ctx *be_ctx;
    void *req_data;
    be_async_callback_t fn;
    void *pvt;
    int restarts;
};
```

Most important from this structure is obviously be_ctx which is described above and pointer req_data. For ID provider this should be cast to struct be_acct_req *. This structure is quite general and can be used for every request, which would be directed to NSS.

In other cases req_data should be cast to struct pam_data*, which includes all information for execution of a PAM request.

Now the function has all needed information to perform query to the server. Results are not directly return to the caller, but data are stored in the database. If data are already in database (from previous searches), they are updated or in the case that server have returned message, that there is not such item, they are deleted from database.

When database is successfully updated, last step is to inform, that everything has been done. This is performed by calling fn function from be_ctx structure. To this function we should provide the result of the query and report if server have responded normally, was offline or a timeout occurred. [2]

Chapter 5

Design

There are three major areas which should be discussed in the design phase.

- Which parts of the provider should be implemented.
- How will be realized the communication with NIS.
- How will be the returned information processed.

5.1 Features to implement

As mentioned every SSSD provider can implement four basic handlers: id, authentication, access and password change. NIS provider must implement id handler, because that is main reason of NIS database and authentication. There is no need for access handler, because NIS maps does not contain any useful information related to access management.

Handling password changes is possible, but this functionality is already provided by `pam_unix2.so`, which can determine if we need to change a password on NIS server through yppasswd protocol.

ID provider should handle requests for users, groups, services and netgroups.

5.1.1 Users

Query for a user means to lookup or enumerate the `passwd` map. This map contains information about users login accounts in the systems. Every value describes single user and contains seven colon-separated fields[5]:

```
name:password:UID:GID:GECOS:directory:shell
```

- Name is the user's login name.
- Password is the encrypted user password.
- UID is the identification number of user
- GID is the numeric primary group ID for this user.
- GECOS is optional and used only for informational purposes. Usually, it contains the full username.

- Directory is the user's home directory: the initial directory where the user is placed after logging in.
- Shell is the program to run at login.

5.1.2 Groups

Query for a group means to lookup or enumerate the `group` map. This map contains information about groups of users in the systems. Every value describes a single group and contains four colon-separated fields[3] :

```
group name:password:GID:user list
```

- Group name is the name of the group.
- Password is the (encrypted) group password. If this field is empty, no password is needed.
- GID is the numeric group ID.
- User list is a list of the usernames that are members of this group, separated by commas.

5.1.3 Services

Query for a service means to lookup or enumerate `services` map. This map is providing a mapping between human-friendly textual names for internet services, and their underlying assigned port numbers and protocol types.[6] Each value describes one service, and is of the form:

```
service-name port/protocol [aliases ...]
```

- Service-name is the friendly name the service is known by and looked up under.
- Port is the port number to use for this service.
- Protocol is the type of protocol to be used. Typical values include tcp and udp.
- Aliases is an optional space or tab separated list of other names for this service.

5.1.4 Netgroups

Query for a netgroup means to lookup in `netgroups` map. This map defines "netgroups", which are sets of triples, used for permission checking when doing remote mounts, remote logins and remote shells.[4] Each line in the file consists of a netgroup name followed by a list of members, where a member is either another netgroup name, or a triple:

```
(host, user, domain)
```

where the host, user, and domain are character strings for the corresponding components. Any of the three fields can be empty, in which case it specifies a "wildcard", or may consist of the string "-" to specify "no valid value". The domain field must either be the local domain name or empty for the netgroup entry to be used.

5.2 Communication with NIS

There are two different way how can we approach to the communication to NIS. Provider can talk directly to NIS servers using RPC calls, or start locally `ypbind` and then use functions from `glibc` to obtain data from database.

5.2.1 RPC communication with NIS server

NIS rpc interfaces provides following procedures:

- YPPROC_NULL is used to check if server is alive.
- YPPROC_DOMAIN checks if `ypserv` serves the named domain.
- YPPROC_DOMAIN_NOACK is the same as YPPROC_DOMAIN, but server does not send ACK when it is not serving the domain. This is used mainly for broadcasts.
- YPPROC_MATCH performs a key lookup.
- YPPROC_FIRST returns first key/value pair from map.
- YPPROC_NEXT returns next key/value pair from map.
- YPPROC_XFR tells server to check for new version of map on master server.
- YPPROC_CLEAR tells `ypserv` to flush it's file cache.
- YPPROC_ALL is used to obtain whole map from server.
- YPPROC_MASTER return master server for domain.
- YPPROC_ORDER returns the order number for a map.
- YPPROC_MAPLIST reeturns list of maps for domain.

Biggest advantage of direct communication with NIS server is no dependence on any other components. But this method has one important drawback. Glibc have a good support for NIS, but it requires, that `ypbind` is running on client a machine. Unfortunately we must assume that users in NIS environment can use applications which are build with NIS support.

5.2.2 Communication through ypbind

Glibc offers following interface which can be used when a machine is running ypbind daemon[7]

```
#include <rpc/rpc.h>
#include <rpcsvc/ypclnt.h>
#include <rpcsvc/yp_prot.h>


int yp_all(char *indomain, char *inmap, struct ypall_callback *incallback);

int yp_bind(char *dom);

int yp_first(char *indomain, char *inmap, char **outkey, int *outkeylen,
             char **outval, int *outvallen);

int yp_get_default_domain(char **domp);

int yp_master(char *indomain, char *inmap, char **outname);

int yp_match(char *indomain, char *inmap, const char *inkey, int inkeylen,
             char **outval, int *outvallen);

int yp_next(char *indomain, char *inmap, char *inkey, int inkeylen,
            char **outkey, int *outkeylen, char **outval, int *outvallen);

int yp_order(char *indomain, char *inmap, char *outorder);

void yp_unbind(char *dom);

char *yperr_string(int incode);

int ypprot_err(unsigned int incode);
```

- `yp_match` provides the value associated with the given key.
- `yp_first` provides the first key-value pair from the given map in the named domain.
- `yp_next` provides the next key-value pair in the given map. To obtain the second pair, the inkey value should be the outkey value provided by the initial call to yp_first. In the general case, the next key-value pair may be obtained by using the outkey value from the previous call to `yp_next` as the value inkey.
- `yp_all` provides a way to transfer an entire map from the server to the client process with a single request. This transfer uses TCP, unlike all other functions in the ypclnt suite, which use UDP. The entire transaction occurs in a single RPC request-response. The third argument to this function provides a way to supply the name of a function to process each key-value pair in the map. `yp_all` returns after the entire transaction

is complete, or the each function decides that it does not want any more key-value pairs.

- `yp_order` returns the order number of a map.
- `yp_master` returns the hostname of the master server for a map.
- `yperr_string` returns a pointer to a null-terminated error string.
- `ypprot_err` converts a YP protocol error code to a ypclnt error code `yperr_string`.

Beside the friendly interface and system-wide accessibility this approach simplifies the communication with multiple NIS servers. `ypbind` will seamlessly switch between servers in case of their unavailability and offers broadcast search for servers. For these reasons designed provider will communicate with NIS server through `ypbind`.

5.3 Handling of requests

SSSD is passing requests to provider by calling function which were specified in the initialization phase. All requests are performed asynchronously in one thread and after fetching data from server, provider should save desired information to cache and by calling callback pass the return state to SSSD.

Unfortunately all yp_* call are blocking, so they can't be called directly in the provider. Due to this reason, the provider should read a request from SSSD, prepare a query for server and then fork a child process which will call the appropriate function, wait for the result and then write them back to the parent process. These data should be processed and store in sysdb.

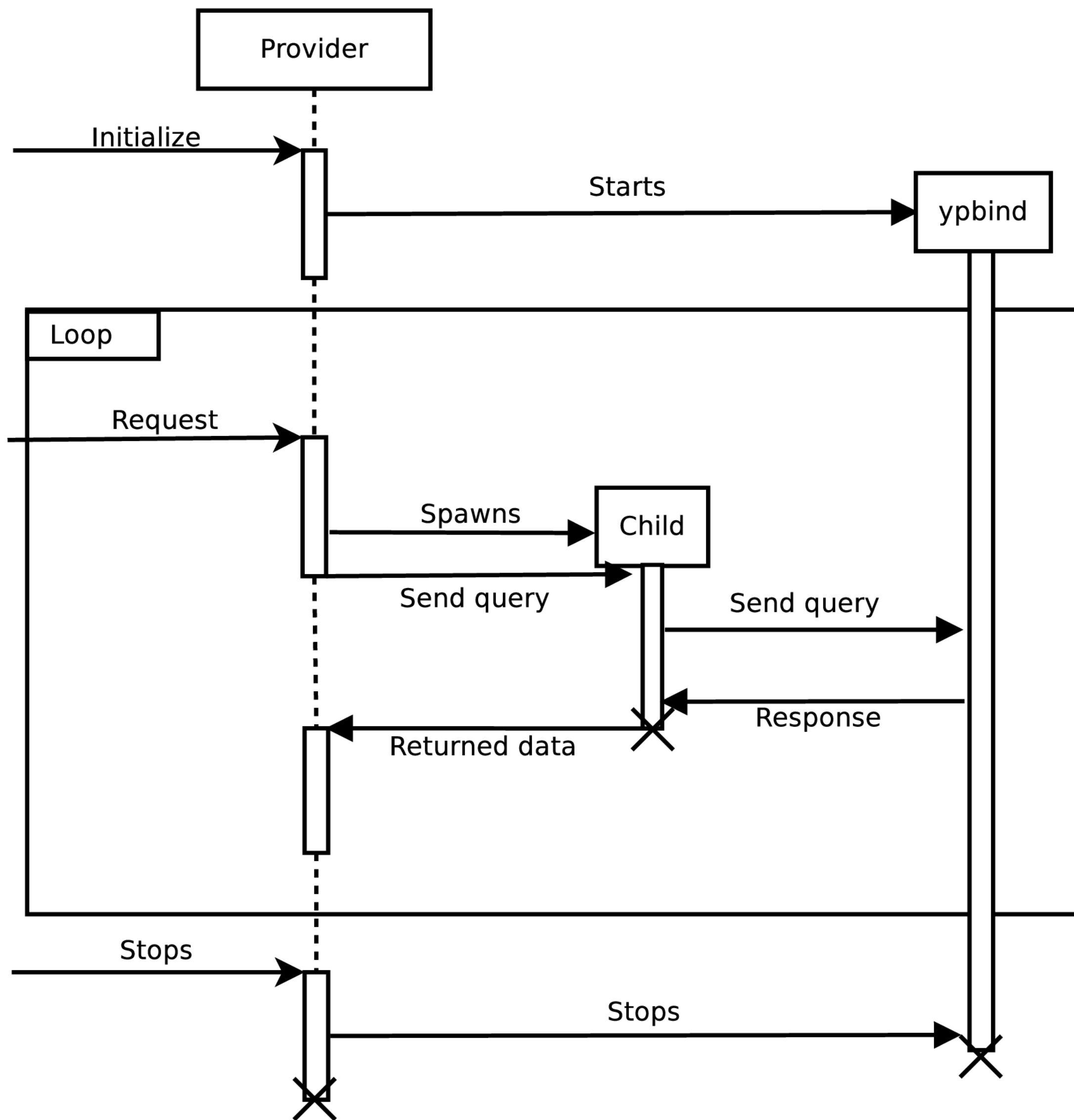


Figure 5.1: Communication in the provider

Chapter 6

Implementation

Implementation can be divided to following parts:

- Initialization
- Handling of ID requests
- Handling of authentication requests
- Communication between parent process and child
- Storing of values

6.1 Initialization

Initialization of provider is handled by two init functions. For initialization of the ID handler SSSD is supposed to call:

```
int sssm_nis_id_init(struct be_ctx *bectx,
        struct bet_ops **ops,
        void **pvt_data);
```

And similarly for the authentication provider:

```
int sssm_nis_auth_init(struct be_ctx *bectx,
        struct bet_ops **ops,
        void **pvt_data);
```

These functions perform setup and returns pointer to handlers functions. Because internally there is no difference between initialization of id and authenticate handler, `sssm_nis_auth_init` is just calling `sssm_nis_id_init` and only modifies handler functions.

Both of these handlers are also sharing the same structure with information called context.

```
struct nis_ctx {
        struct be_ctx *be;
        struct dp_option *nis_opts;
        pid_t ypbind_pid;
        char *domain;
        };
```

This context contains a pointer to general provider structure, the configuration given by user, pid of `ypbind` process and the name of NIS domain. All of these are necessary during whole run of this provider.

Whole initialization process peforms following actions:

1. Allocate memory for context structure
2. Store configuration settings to context
3. Set system domainname with `int setdomainname(const char *name, size_t len);`
4. Write configuration file
5. Start `ypbind` and store its pid to context struct

First the initialization checks SSSD configuration file. User can specify following option:

- nis_domain – name of domain
- nis_server – comma-separated list of servers
- nis_broadcast – use broadcast to find NIS servers
- nis_timeout – timeout for queries to NIS database

Domain must be set by user and he must also specify at least one server or set broadcast search. Broadcast is unset by default and timeout is by default set to 180 seconds.

Initialization continues with writing configuration file for `ypbind`. That means to write into `/etc/ypbyind.conf` line "**domain** *nisdomain* **server** *hostname*" for every server in configuration and "**domain broadcast**" if broadcast search was required.

After configuration files is set, provider can start `ypbind`. Provides forks a new process and the child call `execle`. Timeout for queries is set by environment value `TIMEOUT`.

At this time last thing is set handlers for queries and provider is ready to accept requests.

6.2 Authentication requests

There are multiple pam action, which can SSSD sent to a provider. As mention before designed NIS provider is only able handle simple authentication request.

Handler proceeds with following steps:

1. Allocate context structure for performed action.

   ```
   struct nis_pam_auth_state {
       struct be_req *breq;
       struct pam_data *pd;
       const char *username;
       struct dp_opt_blob password;
   };
   ```

2. Get user login from request and ask NIS database for entry from passwd map.

3. Create copy of given password and set a destructor for it, which will at the end overwrite it, so it does not stays in memory. For this we can use existing function from SSSD.

   ```
   int password_destructor(void *memctx);
   ```

   This function will overwrite password wit null bytes.

4. Check if it got answer from server.

5. Parse accepted line from server.

6. Check if password on server was encoded by supported hash.

7. Hash password from pam request with function

   ```
   int s3crypt_sha512(TALLOC_CTX *memctx,
               const char *key,
               const char *salt,
               char **_hash);
   ```

   Salt is taken from password field of returned value from NIS server.

8. Compare hashes and if they match report success.

9. If requested, store password to sysdb with function

   ```
   int sysdb_cache_password(struct sysdb_ctx *sysdb,
                            const char *username,
                            const char *password);
   ```

Unfortunately in SSSD there is only the SHA512 hash function, so if a server stores passwords in any other type, authentication through SSSD would not work.

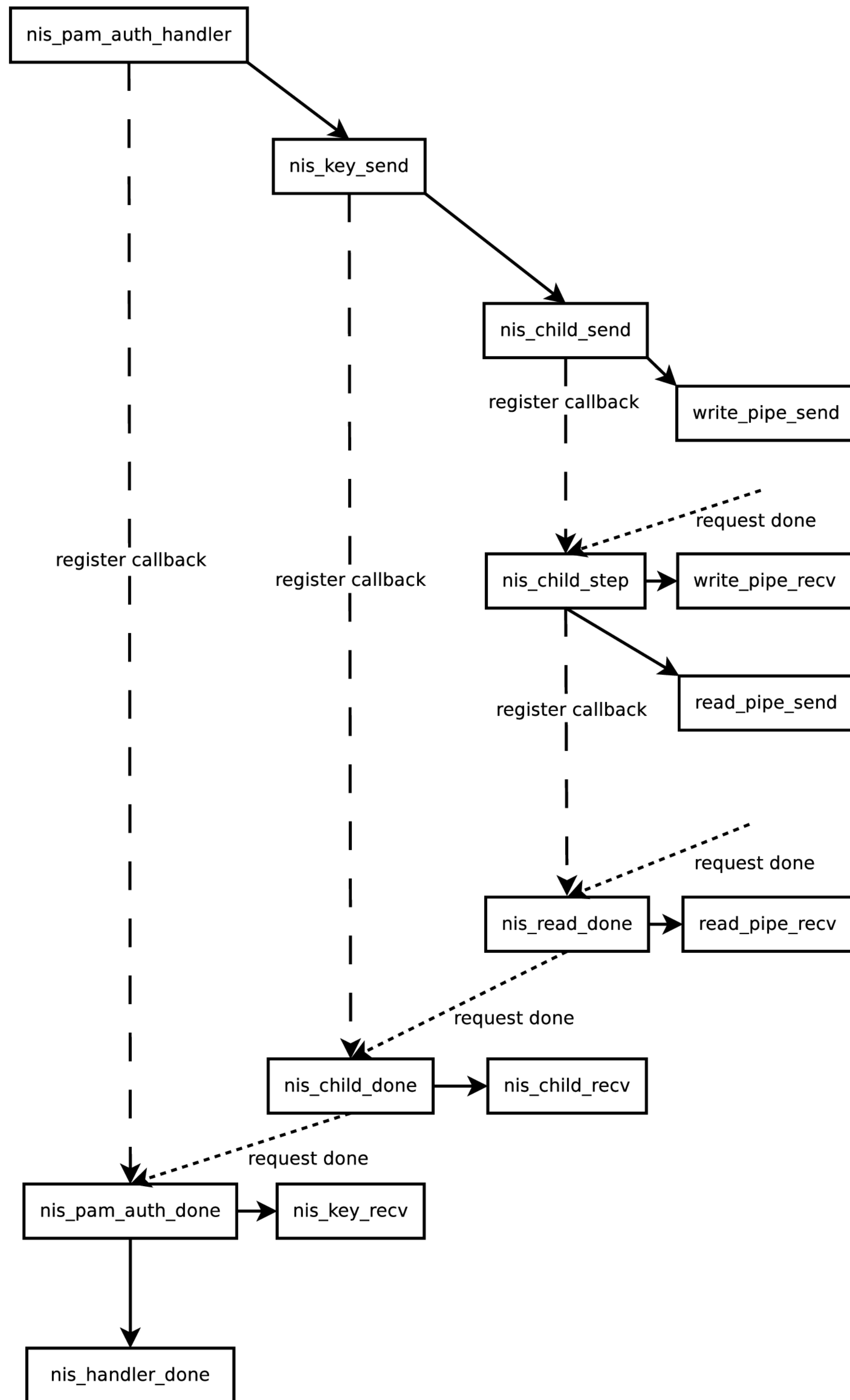


Figure 6.1: Authentication request

6.3 ID requests

Besides netgroups where is possible only to request for context of netgroup, SSSD can ask for lookout of a single value or enumerate the whole table.

After the handler is called, it determines what type of a request is it and if it is the request for lookup or enumeration. For the lookup queries the searched value can be specified as a name (login for user, name of group, name of service) or by its id (uid for user, gid for group, port for service).

The request is then send to the NIS server and after a return found values are stored in the sysdb.

6.3.1 User

Data returned from NIS server are same as in `/etc/passwd`:

```
name:password:UID:GID:GECOS:directory:shell
```

This line is split by colons and values are stored in sysdb. If user was already in sysdb, previous entry is deleted.

In case that requested user was not found in NIS database, value is also removed from sysdb.

There is a set of functions for more abstract work with sysdb.

```
errno_t nis_save_user(struct sss_domain_info *dominfo,
                      struct sysdb_ctx *sysdb, char *response);
errno_t nis_delete_user_name(struct sysdb_ctx *sysdb, char *name);
errno_t nis_delete_user_uid(struct sysdb_ctx *sysdb, uid_t uid);
errno_t nis_save_users(struct sss_domain_info *dominfo,
                       struct sysdb_ctx *sysdb,
                       ssize_t num,
                       char **response);
```

6.3.2 Group

Storing groups is quite similar to users. From NIS server we get line in format of `/etc/group`:

```
group_name:password:GID:user_list
```

Again value is split by colons and store in sysdb. Previous value is deleted.

Only difference here is list of users, this list is parsed and every user is searched in sysdb, if the user exists, it is marked as member of the group. If the user does not exists, a ghost user without no attributes is created and it is included in the group.

Again we have a set of function to operate on sysdb.

```
errno_t nis_save_group(struct sss_domain_info *dominfo,
                       struct sysdb_ctx *sysdb, char *response);
errno_t nis_delete_group_name(struct sysdb_ctx *sysdb, char *name);
errno_t nis_delete_group_gid(struct sysdb_ctx *sysdb, uid_t uid);
errno_t nis_save_groups(struct sss_domain_info *dominfo,
                        struct sysdb_ctx *sysdb,
                        ssize_t num,
                        char **response);
```

6.3.3 Service

Searching for services is again similar to users and groups. There is just one difference in the request. When handler gets request for searching service by a port, it also needs its protocol (usually tcp or upd). Returned values are similar to configuration file `/etc/services`:

```
service-name port/protocol [aliases ...]
```

This line is parsed by spaces and values are stored in sysdb.

Functions to modify sysdb are very simillar to previouse.

```
errno_t nis_delete_service_name(struct sysdb_ctx *sysdb, char *name);
errno_t nis_delete_service_port(struct sysdb_ctx *sysdb, int port, char *proto);
errno_t nis_save_service(struct sss_domain_info *dominfo,
                         struct sysdb_ctx *sysdb,
                         char *response);
errno_t nis_save_services(struct sss_domain_info *dominfo,
                          struct sysdb_ctx *sysdb,
                          ssize_t num,
                          char **response);
```

6.3.4 Netgroup

Request for netgroup is basically an enumeration of it content (this will be described later). Because netgroup can contain other netgroups, child process must perform recursive list on netgroup and all its subgroups. Returned list is set of triples

```
(host, user, domain)
```

which is parsed and values are stored to sysdb. If netgroup already exists in sysdb it is deleted.

Because there are no enumeration requests over netgroup sysdb, we just need only two functions.

```
errno_t nis_delete_netgroup(struct sysdb_ctx *sysdb, char *name);
errno_t nis_save_netgroup(struct sss_domain_info *dominfo,
                          struct sysdb_ctx *sysdb,
                          char *name,
                          ssize_t size,
                          char **response);
```

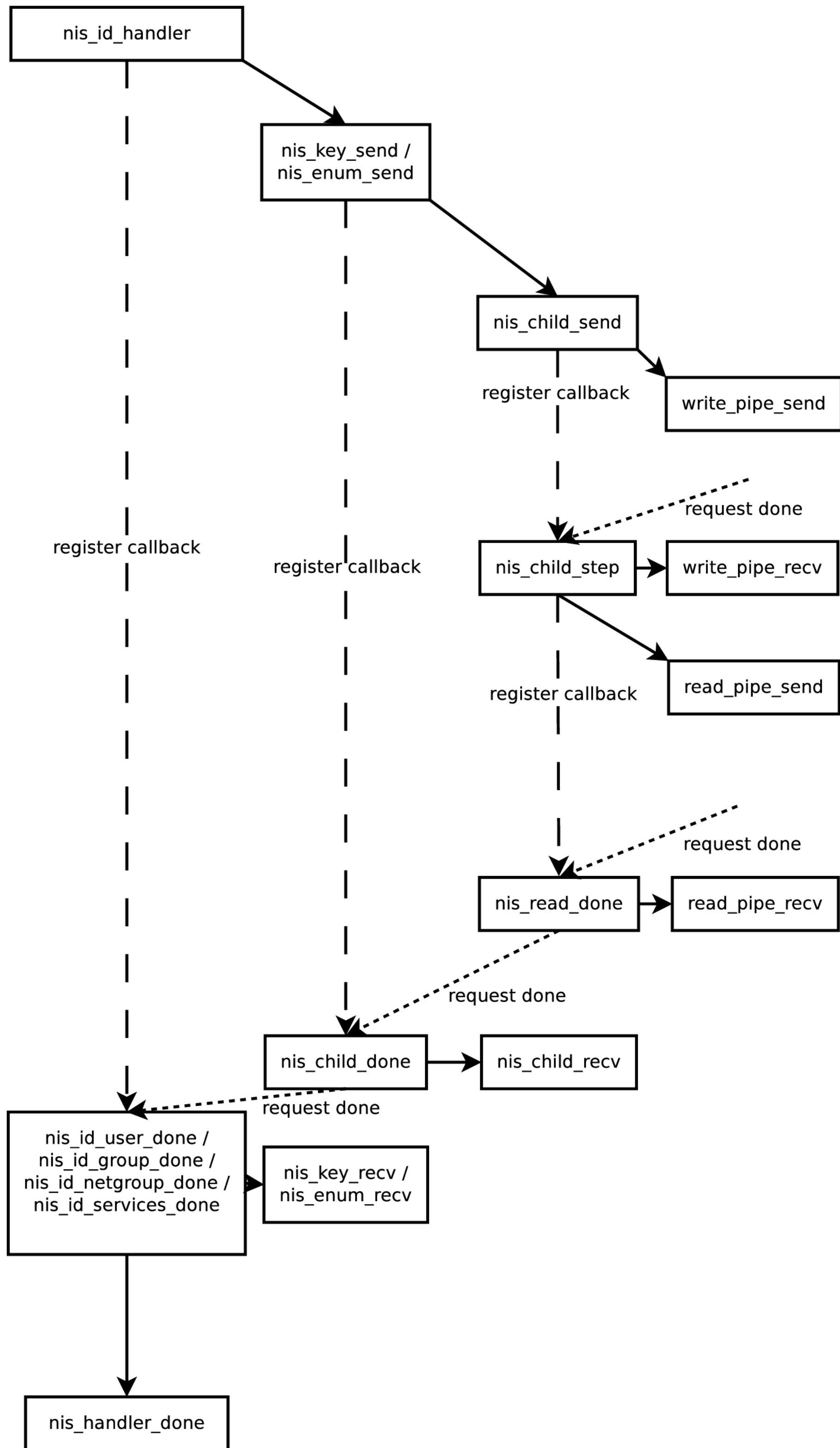


Figure 6.2: ID request

6.4 Communication with NIS

As mentioned before, when provider wants to get some information from NIS, it needs to do it asynchronously.

Sending request to NIS database proceeds following steps:

1. Provider prepares packed buffer with request to child.
2. Provider spawns and setups child.
3. Provider send buffered data to child.
4. Child receives data, unpacks them.
5. Child performs request to NIS database.
6. Child checks return of the requests a prepare packed buffer with answer.
7. Child sends data back to provider and ends.
8. Provider reads data and unpack them.

6.4.1 Preparation of the request

Communication with the child process

For more abstract communication with NIS database through child there are two sets of functions.

- Enumeration

```
errno_t nis_enum_recv(TALLOC_CTX *mem_ctx,
                      struct tevent_req *req,
                      ssize_t *num,
                      char ***data);

struct tevent_req *nis_enum_send(TALLOC_CTX *mem_ctx,
                                 struct tevent_context *ev,
                                 struct nis_ctx *ctx,
                                 const char *map);
```

- Key lookup

```
errno_t nis_key_recv(TALLOC_CTX *mem_ctx,
                     struct tevent_req *req,
                     char **data);

struct tevent_req *nis_key_send(TALLOC_CTX *mem_ctx,
                                struct tevent_context *ev,
                                struct nis_ctx *ctx,
                                const char *map,
                                const char *key);
```

Data packing

After identifying the request from SSSD, the provider must prepare a packed buffer of data, which will be sent to child.

For this purpose, there are two functions.

```
static errno_t nis_key_buf(TALLOC_CTX *mem_ctx,
                           const char *map,
                           const char *key,
                           struct io_buffer **io_buf)
```

This function produces following message:

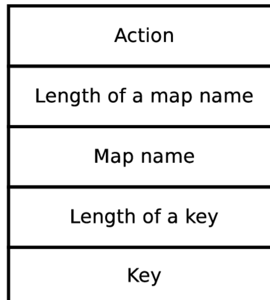


Figure 6.3: Key buffer

```
static errno_t nis_enum_buf(TALLOC_CTX *mem_ctx,
                            const char *map,
                            struct io_buffer **io_buf)
```

This function produces the following message:

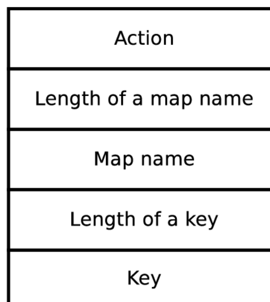


Figure 6.4: Enum buffer

`nis_enum_buf` is used for enumeration requests and `nis_key_buf` for lookups. Both function creates align buffer, where first item is action.

```
enum {
        NIS_ACTION_NONE = 0,
        NIS_ACTION_KEY,
        NIS_ACTION_ENUM,
        NIS_ACTION_NETGROUP
};
```

Action is followed by a length of a name of a map, followed by a name of map. In case of the lookup request, there is appended a length of a key and a key.

6.4.2 Spawning child

Provider spawns a child process in multiple steps:

1. Alocate structures which will store information about a child and a request.

   ```
   struct nis_child {
               pid_t pid;
               int read_from_child_fd;
               int write_to_child_fd;
   };

   struct nis_child_state {
               struct tevent_context *ev;
               struct nis_child *child;
               ssize_t len;
               uint8_t *buf;
   };
   ```

2. Setup a destructor, which will free these structures and clean after child.
3. Create pipes, which will serve for communication with child.
4. Fork new process.
5. In child replace `stdin` and `stdout` with pipes, by use `dup2` and exec child binary.
6. Setup callback functions for reading from child.
7. Setup timeout for child. This timeout is bigger than user specified timeout and is used only in the case of unpredictable behavior of child or `ypbind`.
8. Send buffer to child.

After these steps are done, handler ends and waits for the call of callback function of timeout or pipe handler.

6.5 Child process and queries to NIS

Child process executes following steps:

1. Setup debugging parameters.
2. Get domainname from system by using
   `int getdomainname(char *name, size_t len);`.
3. Read and parse data from input.
4. Determine which action needs to be executed.
5. Perform requested action.
6. Return state and data.

6.5.1 Initialization of child

After child is stated it must setup debugging environment which is configured by SSSD.

Because name of domain is not include in data from master, child will obtain this information directly from system.

Then child can finally read data which was sent from master and parse them. First an action is read and by its value child expect map in case of enumerate request or map and key in case of lookup request or query for netgroups.

All os these values are stored in child context:

```
struct nis_child_status {
    int action;
    char domain[DOMAIN_LEN];
    char *map;
    char *key;
    void *data;
};
```

6.5.2 Query to NIS database

As mentioned child can perform three actions with NIS database.

Key lookup

Key lookup means simply to pass domain, map and key to function `yp_match`.

Enumeration

Enumeration request is performed by calling `yp_all`. One of the arguments passed to this function is a callback, which will be called for every key/value in database. This callback will only save all value to an array.

Content of netgroup

Function will ask NIS server for contain of a netgroup thought `yp_match`, but returned data is already parsed in child, and triples and subgroups are stored in separated arrays. Then this function is called recursively for every subgroup. Before each call name of subgroup is compared to list of already browsed groups to eliminate potential loops. [1]

6.5.3 Return of values from child

Every `yp_*` function can return one of this return codes:

- `YPERR_ACCESS` – Access violation.
- `YPERR_BADARGS` – The arguments to the function are bad.
- `YPERR_BADDB` – The YP database is bad.
- `YPERR_BUSY` – The database is busy.
- `YPERR_DOMAIN` – Cannot bind to server on this domain.
- `YPERR_KEY` – No such key in map.
- `YPERR_MAP` – No such map in server's domain.
- `YPERR_NODOM` – Local domain name not set.
- `YPERR_NOMORE` – No more records in map database.
- `YPERR_PMAP` – Cannot communicate with rpcbind.
- `YPERR_RESRC` – Resource allocation failure.
- `YPERR_RPC` – RPC failure; domain has been unbound.
- `YPERR_YPBIND` – Cannot communicate with `ypbind`.
- `YPERR_YPERR` – Internal YP server or client error.

`YPERR_SUCCESS` is interpreted as success. `YPERR_KEY` means that key was not found. `YPERR_RPC`, `YPERR_YPERR`, `YPERR_PMAP`,`YPERR_YPBINDYPERR_YPSERV` and `YPERR_BUSY` are probably recoverable errors, so we will report that server is temporally down. Other codes mean fatal error.

[1]It seems that `ypserv` package is not able to handle loops in structure of netgroups https://bugzilla.redhat.com/show_bug.cgi?id=962178.

After the child has requested data, it again creates packed buffer. Return code goes to first position followed by returned data, which are represented as length of value and value itself in case of reply to lookup request.

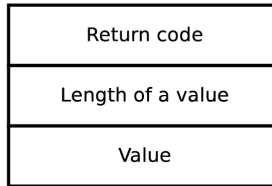


Figure 6.5: Buffer with reply to lookout request

Or followed by multiple values in case of enum request and lookup in netgroup map.

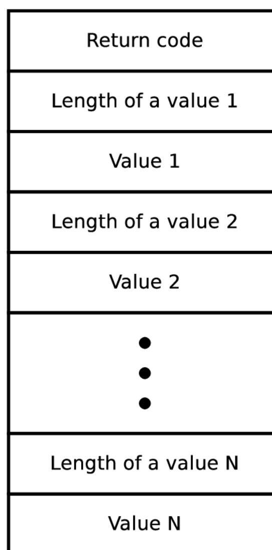


Figure 6.6: Buffer with reply to enum request

Chapter 7

Testing

After the provider was created it is necessary to test its functionality.

7.0.4 ID provider

Testing the ID provider is quite simple. After installing SSSD with the NIS provider we edit `/etc/nsswitch.conf` and modify following lines.

```
passwd:     sss files
group:      sss files
services:   sss files
netgroup:   sss
```

And setup `/etc/sssd/sssd.conf`. For example:

```
[sssd]
domains = nis
services = nss
config_file_version = 2

[domain/nis]
id_provider = nis
nis_domain = ruenix.cz
nis_server = 192.168.122.1, localhost
nis_broadcast = true
```

Then we can try to obtain data by using `getent` command.

7.0.5 Authentication provider

It is not safe to test this provider the same direct way as the ID provider. So for this purpose I have used utility `pamtester`.

We create a new pam configuration file, for example `/etc/pam.d/sss-test`.

```
auth        required      pam_env.so
auth        sufficient    pam_fprintd.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        sufficient    pam_sss.so use_first_pass
```

```
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        required      pam_deny.so
```

Then we configure sssd to also process pam requests.

```
[sssd]
domains = nis
services = nss,pam
config_file_version = 2

[domain/nis]
id_provider = nis
auth_provider = nis
nis_domain = ruenix.cz
nis_server = 192.168.122.1, localhost
nis_broadcast = true
enumerate = true
```

Now we can test the authentication provider by calling `pamtester sss-test user operation`.

Chapter 8

Conclusion

Despite the fact that NIS belongs more or less between outdated technologies and in most ways it can be replaced by LDAP, there are still networks which are using NIS and due to easy setup NIS can still find new users.

Designed provider should cover all possible demands for identification and authorization. Additionally it offers possibility to cache credentials in the SSSD cache, which will be a huge benefit in networks where NIS is used and users have laptops which sometimes disconnect from a network. Moreover provider has a very simple configuration.

There are two main areas, where this project can be enhanced. Biggest weakness of this provider is limitation to SHA512 hashing function for passwords. This could be solved by using a crypto library or implementing other hash function to SSSD.

Second area for improvement is to add a possibility to cache other maps then user, group, services and netgroup, for example hosts.

Bibliography

[1] Ldap concepts & overview. http://www.zytrax.com/books/ldap/ch2/, 2011.

[2] Sssd – system security services daemon. https://fedorahosted.org/sssd/, 2011.

[3] group(5) – linux man page. http://linux.die.net/man/5/group, 2013.

[4] netgroup(5) – linux man page. http://linux.die.net/man/5/netgroup, 2013.

[5] passwd(5) – linux man page. http://linux.die.net/man/5/passwd, 2013.

[6] services(5) – linux man page. http://linux.die.net/man/5/services, 2013.

[7] ypclnt(3) – bsd library functions manual. http://www.manpagez.com/man/3/ypclnt/, 2013.

[8] K. Benák. Použití adresářových služeb v informačních systémech. Master's thesis, České vysoké učení technické v Praze, 2004.

[9] S. Graham, S. Shah, and J. Hynek. *Administrace systému Linux: podrobný průvodce začínajícího administrátora*. Grada Publishing, 2003.

[10] R.J. Hontañón and H.L. Roubíček. *Linux - praktická bezpečnost*. Grada Publishing, 2003.

[11] J. Hradílek, D. Silas, M. Prpič, E. Kopalová, E. Slobodová, J. Ha, D. O'Brien, M. Hideo, and D. Domingo. Fedora 15 deployment guide. http://docs.fedoraproject.org/en-US/Fedora/15/html/Deployment_Guide/index.html, 2011.

[12] J. Hrozek. Freeipa a sssd – pokročilá správa uživatelů v linuxu. http://jhrozek.fedorapeople.org/sssd.pdf, 2010.

[13] T. Kukuk. The linux nis(yp)/nys/nis+ howto. http://www.tldp.org/HOWTO/NIS-HOWTO/index.html, June 2003.

[14] Sun Microsystems. System administration guide: Naming and directory services (dns, nis, and ldap). http://docs.oracle.com/cd/E19683-01/817-4843/index.html, 2004.

[15] I.B.M. Redbooks. *IBM E Server Certification Study Guide-AIX 5L Communications*. Vervante, 2004.

[16] H. Stern, M. Eisler, and R. Labiaga. *Managing NFS and NIS*. A nutshell handbook. O'Reilly & Associates, 2001.