Server administration

Remo Suppi Boldrito

PID_00148466

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction

The interconnection between machines and high-speed communications has meant that the resources that are used can be at a different geographical location to that of the user. UNIX (and of course GNU/Linux) is probably the best example of this philosophy, because from its beginning, the focus has always been on the sharing of resources and the independence of the 'devices'. This philosophy has been realized in the creation of something that has now become very common: the services. A service is a resource (which may or not be universal) that makes it possible to obtain information, share data or simply process information remotely, under certain conditions. Our objective is to analyse the services that make it possible for our network. Generally, within a network, there will be a machine (or various machines, depending on the configuration) that will make it possible to exchange information with all the other elements. These machines are called servers and they contain a set of programs that centralise the information and make it easily accessible. These services help to reduce costs and increase the availability of information, but it should be remembered that a centralised service also involves some disadvantages, as it can come offline and leave the users without the service.

> The servers should be designed so that all the servers are mirrored to solve these situations.

The services can be classified into two categories: those linking computers to computers or those linking users to computers. In the first case, the services are those needed by other computers, whereas in the second, the services are those required by the users (although there are services that may fall into both categories). In the first category, there are the naming services, such as the domain name system (DNS), the user information service (NISYP), the LDAP information directory or the services for storing in proxies. In the second category, we have interactive connection and remote execution services (SSH, telnet), file transfer (FTP), user-level information exchange such as email (MTA, IMAP, POP), news, World Wide Web, Wiki and files (NFS). To demonstrate all the possibilities of GNU/Linux Debian-FC6, we will describe each of these services with a minimal and operative configuration, but without leaving out the aspects related to security and stability.

1. Domain name system (DNS)

The function of the DNS service (as we explained in the unit on network administration) is to translate the machine names (legible and easy to remember for users) into IP addresses or vice-versa.

Example

When we ask the IP address of pirulo.remix.com is, the server will respond 192.168.0.1 (this process is known as mapping); likewise, when we request an IP address, the service will respond with the name of the machine (known as reverse mapping).

The domain name system (DNS) is a tree architecture that avoids duplicating information and makes any searches easier. For this reason, a single DNS makes no sense unless it is part of the architecture.

The application that provides this service is called *named*, it is included in most GNU/Linux distributions (/usr/sbin/named) and it is part of the package called *bind* (currently version 9.x) coordinated by the ISC (*Internet software consortium*). The DNS is simply a database, which means that the people that modify it have to be aware of its structure, as, otherwise, the service will be affected. As a precaution, special care must be taken to save copies of the files to avoid any interruption in the service. The package in Debian comes as bind and bind.doc. [LN01, Deb03c, IET03]. The configurations are similar, as they are FC, but you will have to install bind, bind-utils and caching-nameserver which will be managed by the yum for example.

1.1. Cache names server

Firstly, we will configure a DNS server to resolve requests, which will act as a cache for name queries (resolver, caching only server). In other words, the first time, the appropriate server will be consulted because we are starting with a database that contains no information, but all subsequent times, the cache names server will respond, with the corresponding decrease in response times. To configure the cache names server, we need the `/etc/bind/named.conf` file (in Debian), which has the following (the original comments within the file, indicated with //, have been respected):

```
options {
directory "/var/cache/bind";
            // query-source address * port 53;
            // forwarders {
            //    0.0.0.0;
            //
```

```
            };
            auth-nxdomain no; # conform to RFC1035
            };
// prime the server with knowledge of the root servers}
zone "." {
            type hint;
            file "/etc/bind/db.root"; };
            // be authoritative for the localhost forward and reverse zones, and for
            // broadcast zones as per RFC 1912
            }
zone "localhost" {
            type master;
            file "/etc/bind/db.local";
            };
zone "127.in-addr.arpa" {
            type master;
            file "/etc/bind/db.127";
            };
zone "0.in-addr.arpa" {
            type master;
            file "/etc/bind/db.0";
            };
zone "255.in-addr.arpa" {
            type master;
            file "/etc/bind/db.255";
            };
// add entries for other zones below here
}
```

The directory sentence indicates where we will find the remaining configuration files (/var/cache/bind in our case). The /etc/bind/db.root file will contain something similar to the following (only the first lines, which are not comments indicated by a ';', are shown, and care must be taken with the dots [.]) at the beginning of some lines –they can be obtained and updated directly from the Internet–):

```
...
; formerly NS.INTERNIC.NET
;
. 3600000 IN NS A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000 A 198.41.0.4
;
; formerly NS1.ISI.EDU
;
. 3600000 NS B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000 A 128.9.0.107
;
; formerly C.PSI.NET
;
. 3600000 NS C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET. 3600000 A 192.33.4.12
;
...
```

This file described the root name servers in the world. These servers change, which means that the file must be updated regularly from the Internet. The following sections are the zones; the localhost and 127.in-addr.arpa zones, that link the files to the `etc/bind/db.local` and `/etc/bind/db.127` directories, refer to the direct and inverse resolution for the local interface. The following zones are for the broadcast zones (see RFC 1912) and the appropriate zones should be added at the end. For example, the db.local file could be (';' means 'comment'):

```
; BIND reverse data file for local loopback interface
$TTL 604800
@   IN   SOA    ns.remix.bogus.   root.remix.bogus. (
         1                        ; Serial
         604800                   ; Refresh
         86400                    ; Retry
         2419200                  ; Expire
         604800)                  ; Negative Cache TTL
@        IN          NS           ns.remix.bogus.
1.0.0    IN          PTR          localhost.
```

We will explain how it is used later. The next step is to put the name server in /etc/resolv.conf:

```
search subdomain.your-domain.domain your-domain.domain
# for example search remix.bogus bogus
nameserver 127.0.0.1
```

Where we will have to replace the subdomain.your-domain.domain with the appropriate values. The search line indicates which domains will be searched for any host that wants to connect (it is possible to replace *search* with *domain*, although they behave differently) and the name server specifies the address of the name server (in this case, your actual machine, which is where the naming process will execute). The search behaves as follows: if a client is searching for the machine called pirulo, first, the pirulo.subdomain.your-domain.domain will be searched, then pirulo.your-domain.domain and finally, pirulo. This means that the search will take some time; however, if pirulo will be in subdomain.your-domain.domain, it is not necessary to enter the rest.

The next step is to start up *named* and look at the results of the execution. To start up the daemon, we can directly use the `/etc/init.d/bind9 start` startup script (if the *named* is already executing, go to `/etc/init.d/bind9 reload`) or, if not, `/usr/sbin/named`. If we look at the system log in /var/log/daemon.log, we will see something similar to:

```
Sep 1 20:42:28 remolix named[165]: starting BIND 9.2.1 \\
Sep 1 20:42:28 remolix named[165]: using 1 CPU \\
Sep 1 20:42:28 remolix named[167]: loading configuration from '/etc/bind/named.conf'
```

The server's startup and the error messages will appear here (if there were any errors, in which case they must be corrected and the process started again). We can now verify the configuration with commands such as nslookup (original and easy but obsolete according to the programmers), host or dig (recommended). The output of dig -x 127.0.0.1 will be something like:

```
# dig -x 127.0.0.1
;; <<>> DiG 9.2.1 <<>> -x 127.0.0.1
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31245
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION: ;1.0.0.127.in-addr.arpa. IN PTR
;;ANSWER SECTION: 1.0.0.127.in-addr.arpa. 604800 IN PTR localhost.
;; AUTHORITY SECTION: 127.in-addr.arpa. 604800 IN NS ns.remix.bogus.
;; Query time: 1 msec
;; SERVER: 127.0.0.1 #53(127.0.0.1)
;; WHEN: Mon Sep 1 22:23:35 2003
;; MSG SIZE rcvd: 91
```

Where we can see that the query has taken 1 millisecond. If you have an Internet connection, you can search for a machine within your domain and see the server performance and behaviour. In BIND9 there is the lwresd (*lightweight resolver daemon*), which is the daemon that provides naming services to clients that use the BIND9 *lightweight resolver* library. It is essentially a cache server (like the one we have configured) that makes the queries using BIND9 *lightweight resolver protocol* instead of the DNS protocol. This server listens through interface 127.0.0.1 (which means it only attends to processes in the local host) in UPD and port 921. The client requests are decrypted and resolved using the DNS protocol. When responses are obtained, the lwresd encodes them in the lightweight format and returns them to the client that has requested them.

Finally, as we have mentioned, the kernel uses various sources of information, which, for the network, are obtained from `/etc/nsswitch.conf`. This file indicates from where we obtain the source of information and there is a section, for machine names and IPs, such as:

```
hosts: files dns
```

This line (if it is not there, it should be added) indicates that whoever needs a machine name or IP should first check `/etc/hosts` and then in DNS, in accordance with the domains indicated in `/etc/resolv.conf`.

1.2. Forwarders

In networks with a large workload, it is possible to balance the traffic using the section on forwarders. If your Internet Service Provider (ISP) has one or more stable name servers, it is advisable to use them to decongest the requests on the server. For this, we must delete the comment (//) from each line in the

forwarders section of the /etc/bind/named.conf file and replace the 0.0.0.0 with the IPs of the name servers of our ISP. This configuration is advisable when the connection is slow, when using a modem, for example.

1.3. Configuration of an own domain

DNS possesses a tree structure and the origin is known as '.' (see `/etc/bind/db.root`). Beneath the '.' there are the TLDs (*top level domains*) such as org, com, edu, net etc. When searching in a server, if the server does not know the answer, the tree will be searched recursively until it is found. Each '.' in an address (for example, pirulo.remix.com) indicates a different branch of the DNS tree and a different scope for requesting (or responsibility) that will be followed recursively from left to right.

Another important aspect, apart from the domain, is in-addr.arpa (*inverse mapping*), which is also nested as the domains and serves to obtain names when requesting by IP address. In this case, the addresses are written the other way round, in accordance with the domain. If pirulo.remix.com is 192.168.0.1, it will be written as 1.0.168.192, in accordance with pirulo.remix.com.

We must then configure the actual remix.bogus domain in file `/etc/bind/db.127` [LN01]:

```
; BIND reverse data file for local loopback interface
$TTL 604800
@ IN SOA ns.remix.bogus. root.remix.bogus. (
      1                  ; Serial
      604800             ; Refresh
      86400              ; Retry
      2419200            ; Expire
      604800 )           ; Negative Cache TTL
@     IN      NS        ns.remix.bogus.
1.0.0 IN      PTR       localhost.
```

The '.' must be taken into account at the end of the domain names. The origin of a zone's hierarchy is specified by the identification of the zone, which in our case, is 127.in-addr.arpa. This file (db.127) contains 3 registries: SOA, NS, PTR. The SOA (*start of authority*) must be in all of the zone files, at the beginning, after TTL and the @ signifies the origin of the domain; NS, the name server for the domain and PTR (*domain name pointer*), which is host 1 in the subnet (127.0.0.1) and is called local host. This is the series 1 file and root@remix.bogus (last space in the SOA line) is in charge of it. We could now restart the *named* as shown above and, using dig -x 127.0.0.1, we could see how it works (identically to that shown previously).

We would then have to add a new zone in named.conf:

```
zone "remix.bogus" {
   type master;
    notify no;
    file "/etc/bind/remix.bogus";
    };
```

We must remember that in named.conf, the domains appear without the '.' at the end. In the file remix.boguswe will put the hosts of which we will be in charge:

```
; Zone file for remix.bogus
$TTL       604800
@          IN       SOA      ns.remix.bogus.  root.remix.bogus. (
           199802151         ; serial, todays date + todays serial
           604800            ; Refresh
           86400             ; Retry
           2419200           ; Expire
           604800 )          ; Negative Cache TTL
@          NS       ns       ; Inet Address of name server
           MX       10       mail.remix.bogus.  ; Primary Mail Exchanger
localhost           A        127.0.0.1
ns                  A        192.168.1.2
mail                A        192.168.1.4
                    TXT "Mail Server"
ftp                 A        192.168.1.5
                    MX       10 mail
www                 CNAME    ftp
```

A new MX registry, the Mail exchanger, appears here. This is the place to which the emails that arrive will be sent, someone@remix.bogus, and they will be sent to mail.remix.bogus (the number indicates the priority if we have more than one MX). Always remember the '.' that is necessary in the zone files at the end of the domain (if these are not entered, the system will add the SOA domain at the end, which would transform mail.remix.bogus, for example, into `mail.remix.bogus.remix.bogus`, which would be incorrect). CNAME (*canonical name*) is used to give a machine one alias or various aliases. As of this moment, we would be able (after the `/etc/init.d/bind9 reload`) to test, for example, `dig www.remix.bogus`.

The last step is to configure the inverse zone, in other words, so that IP addresses can be changed into names, for example, adding a new zone:

```
zone "192.168.1.in-addr.arpa" {
   type master;
   notify no;
   file "/etc/bind/192.168.1";
   };
```

And the file /etc/bind/192.168.1 similar to the preceding one:

```
$TTL 604800
```

```
@    IN    SOA       ns.remix.bogus. root.remix.bogus. (
     199802151       ; serial, todays date + todays serial
     604800          ; Refresh
     86400           ; Retry
     2419200         ; Expire
     604800 )        ; Negative Cache TTL
@    NS              ns.remix.bogus.
2    PTR             ns.remix.bogus
4    PTR             mail.remix.bogus
5    PTR             ftp.remix.bogus
```

This can be tested again with `dig -x 192.168.1.4`. We must remember that these examples are on private IPs, in other words, not on Internet IPs. Another important point is that we must not forget the notify no, as otherwise, our experiments with the DNS will spread to the servers through the DNS tree (possibly even modifying the DNS of our provider or institution). These should only be modified when we are sure that it works and we are certain about the changes we want to make. To look at a real example, please see DNS-HOWTO at http://tldp.org/HOWTO/DNS-HOWTO-7.html.

Once we have created a master server, we must create a slave server for security, which is identical to the master, except in that the zone in the place of the type master must have a slave and the IP of the master. For example:

```
zone "remix.bogus" {
    type slave;
    notify no;
    masters {192.168.1.2; }
    };
```

2. NIS (YP)

In order to facilitate the administration and make the system more user-friendly, in networks of different sizes that execute GNU/Linux (or Sun's Solaris or any other operating system that supports this service), there are Network Information Services, NIS (or yellow pages, YP, Sun's original name). GNU/Linux can provide support as an NIS client/server and can act as a client ("beta" version) of NIS+, which is a safer and more optimised version of NIS. The information that can be distributed in NIS is: users (login names), passwords (`/etc/passwd`), user directories (home directories), group information (`/etc/group`), which has the advantage that, from any client machine or from the server itself, the user may connect with the same account and password and to the same directory (although the directory must have been previously mounted on all the machines with NFS or using the automount service). [Miq01, Kuk03]

The NIS architecture is of the client-server type, in other words, there is a server that will have all the databases and some clients that will consult these data in a transparent manner for the user. For this reason, we must consider configuring the 'reinforcement' servers (called secondary servers) so that users will not be blocked because the primary server is unavailable. This architecture is called multiple server architecture (master+mirrors-clients).

2.1. ¿How do we initiate a local NIS client in Debian?

Having a local client means adding the computer to an existing NIS domain:

- First, we must verify that the *netbase* (basic TCP/IP network), *portmap* (server that turns RPC numbers into DARPA ports and that is necessary for programs that execute RPC, including NFS and NIS) and *nis* (specific) packages have been installed. We recommend using the kpackage command or directly with apt-get (we can check if this is installed with `apt-cache pkgnames`) in text mode. When installing the NIS package, you will be prompted for a domain (NIS *domainname*). This is a name that will describe the set of machines that will use NIS (it is not a host name). Bear in mind that NISPirulo is different to Nispirulo as a domain name. To configure this domain, you may use the command `nisdomainname`, a domain which will be saved in /proc/sys/kernel/domainname.

- Firstly, we must start up the portmap service with:
  - `/etc/init.d/portmap start`

  - We can check whether these are active using `rpcinfo -p`.

- If the NIS server is not local, we must use the ypbind command. The ypbind command is used to find a server for the specified domain, whether it is through a broadcast (not recommended, as it is not secure) or through searching the server indicated in the configuration file /etc/yp.conf (recommended). The /etc/yp.conf file has the following syntax:
  - *domain nisdomain server hostname*: indicates that the *hostname* should be used for the *nisdomain*. It is possible to have more than one entry of this type for the same domain.

  - *domain nisdomain broadcast*: indicates that *broadcast* should be used on the local network to discover a server with a *nisdomain*.

  - *ypserver hostname*: indicates that *hostname* should be used as a server. It is advisable to use this line (*ypserver*) where we must enter the IP address of the NIS server. If the name is specified, make sure that the IP can be found by DNS or that it appears in the /etc/hosts file, as, otherwise, the client will be blocked.

- Start up the service by executing:
  - `/etc/init.d/nis stop`

  and then:
  - `/etc/init.d/nis start`

- This steps must start up, the NIS client will be working (this can be confirmed with `rpcinfo` or `localhost ypbind`, which will show the two versions of the active protocol) or we can use the `ypcat mapname` command (for example, `ypcat passwd`, which will show the NIS users defined in the server) where the relationship of the *mapnames* to the tables in the NIS database are defined in /var/yp/nicknames.

2.2. What resources must be specified in order to use NIS?

Let us assume that we have installed one of Debian's latest distributions (for example, 3.0 Woody or 3.1 *Sarge*), which supports Libc6 (the same for FC4 or higher) and that we want to configure the system so that the users of one client machine may access the information in the server. In this case, we must send the *login* request to the appropriate databases by:

1) Verify that the /etc/nsswitch.conf and ensure that the passwd, group, shadow and netgroup entries are similar to:

passwd: compat
group: compat

shadow: compat ...
netgroup: nis

See *man nsswitch.conf* for the syntax of this file.

2) Add the following line in the NIS client machines, in the /etc/passwd file at the end of the file (this will indicate that if the user is not local, the NIS server will be asked):

+:::::: (one '+' and six ':')

3) It should be remembered that in /etc/passwd we can use the + and the **?** in front of each user name in /etc/passwd, to include/override the login of these users. If we are using passwords with shadows (more secure, as it will not allow a normal user to see the encrypted passwords of other users), the following line must be added at the end of the /etc/shadow file:

+:::::::: (one '+' and eight ':')

4) The following line must also be added at the end of /etc/group:

+::: (one '+' and three ':')

5) The searches for hosts (host lookups) will be carried out through DNS (and not NIS), which means that, for Libc6 applications, in file /etc/nsswitch.conf we will have to change the hosts entry for the following line: hosts: files dns. Or, if we prefer to do this using NIS, `hosts: files nis.` For Libc5 applications, we must modify the /host.conf file by entering order hosts, DNS or order hosts, NIS, as required.

> With this configuration, it will be possible to establish a local connection (over the NIS client) with a user that is not defined in the /etc/passwd file, in other words, a user defined in another machine (ypserver).

For example, we may execute `ssh -l user localhost`, where the user is a user defined in ypserver.

2.3. How should we execute a master NIS server?

Let us assume that we have installed the nis package and portmap (and portmap is working) and that the data bases of the NIS have been created (see the following section):

- We must make sure that /etc/hosts contains all the machines that will form part of the domain in the FQDN (*fully qualified domain name*)

format, which is where the IP, the name and domain and the name without the domain of each machine appears (for example, 192.168.0.1 pirulo.remix.com pirulo). This is only necessary in the server, as the NIS does not use DNS.

- In addition, it exists in the /etc/defaultdomain file with the chosen domain name. Do not use your DNS domain so that you do not incur in any security risks, unless you appropriately configure the files /etc/ypserv.securenets, which indicates the sites from which the clients will be able to connect with a netmask/network pair, and /etc/ypserv.conf, which carried out a more detailed control because it indicates which hosts can access which maps, for example: passwd.byname o shadow.byname.

- Verify that *NISSERVER* = *master* exists in /etc/default/nis.

- For security reasons, it is possible to add the local network number to the /etc/ypserv.securenets file.

- Start up the server executing the `/etc/init.d/nis stop` command and then the `/etc/init.d/nis start` command. This command will start up the server (ypserv) and the password daemon (yppasswdd), which may be consulted if it is active with `ypwich -d domain`.

2.4. How should we configure a server?

The server is configured with the command `/usr/lib/yp/ypinit -m`; however, it is necessary to verify that the `/etc/networks` file exists, as it is essential for this script.

If this file does not exist, create an empty one with `touch /etc/networks`. It is also possible to make the ypbind client execute on the server; in this way, all the users entered by NIS, as mentioned above, modifying the /etc/passwd file, where all the normal entries before the line +:::::: will be ignored by the NIS (they may only access locally), whereas the subsequent ones may access through the NIS from any client. [Miq01]

Consider that as of this moment, the commands for changing the passwd or user information such as passwd, chfn, adduser are no longer valid. Instead, we will have to use commands such as yppasswd, ypchsh and ypchfn. If we change the users or modify the abovementioned files, we will have to rebuild the NIS tables by executing the make command in the /var/yp directory to update the tables.

Bear in mind that the Libc5 does not support shadow passwords, which means that shadow should not be used with NIS, if we have applications with Libc5. There will not be a problem if we have Libc6, which accepts NIS with shadow support.

Configuring a slave server is similar to configuring a master server, except in that *NISSERVER = slave* in /etc/default/nis. On the master, we must indicate that it has to distribute the tables automatically to the slaves by entering *NOPUSH = "false"* in the /var/yp/Makefile file.

Now we must tell the master who its slave is, by executing:

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

and entering the names of the slave servers. This will rebuild the maps but it will not send the files to the slaves. For this, on the slave, execute:

```
/etc/init.d/nis stop
```

```
/etc/init.d/nis start
```

and, finally:

```
/usr/lib/yp/ypinit -s name_master_server.
```

In this way, the slave will load the tables from the master.

It would also be possible to place the nis file in the /etc/cron.d directory with a content similar to (remember to perform an chmod 755 /etc/cron.d/nis):

```
20 * * * * root /usr/lib/yp/ypxfr_1perhour >/dev/null 2>&1
40 6 * * * root /usr/lib/yp/ypxfr_1perday >/dev/null 2>&1
55 6,18 * * * root /usr/lib/yp/ypxfr_2perday >/dev/null 2>&1
```

With which we will ensure that all the changes in the master will be transferred to the slave NIS server.

Recommendation: After using `adduser` to add a new user on the server execute `make -C /var/yp` to update the NIS tables (and do this whenever any user characteristic changes, for example, the password with the passwd command, which will only change the local password and not the NIS password). To check that the system is working and that the user is registered in the NIS, you may execute `ypmatch userid passwd` where userid is the user previously registered with adduser and after performing the *make*. To verify that the NIS system is working, you may use the script of http://tldp.org/HOWTO/NIS-HOWTO/verification.html, which permits a more detailed verification on the NIS.

3. Remote connection services: telnet and ssh

3.1. Telnet and telnetd

Telnet is a (client) command used to communicate interactively with another host that executes the daemon telnetd. The telnet command may be executed as telnet host or interactively as telnet, which will enter the "telnet>" prompt, and then, for example: *open host*. Once communication has been established, we must enter the user and the password with which we wish to connect to the remote system. There are various commands (in the interactive mode), such as open, logout, mode (defines the visualisation characteristics), close, encrypt, quit, set, unset, or you may execute external commands with '!'. You may use the /etc/telnetrc file for default definitions or .telnetrc the definitions of a particular user (these must be in the user's home directory).

The telnetd daemon is the telnet protocol server for the interactive connection. Telned is generally started up by the inetd daemon and it is recommended that a tcpd wrapper (which uses the access rules in host.allow and host.deny) be included in the telnetd call within the /etc/inetd.conf file (for example), include a line such as:

telnet stream tcp nowait telnetd.telenetd /usr/sbin/tcpd /usr/bin/in.telnetd)

To increase the system's security, please see the unit on security. In some distributions (Debian 3.0 or higher), inetd's functions can be replaced by xinetd, which means that the /etc/xinetd.conf file must be configured (see the unit on the network administration). Likewise, if you wish to start up inetd in test mode, you can use the sentence `/etc/init.d/inetd.real start`. If the /etc/uissue.net file is present, telnetd will show its contents when logging in. It is also possible to use /etc/security/access.conf to enable/disable user logins, host logins or user group logins, as they connect.

It should be remembered that, although the *telnet-telnetd* pair may function in encrypt mode in the latest versions (transfer of encrypted data, although they must be compiled with the corresponding option), it is an absolete command (deprecated), mainly due to the lack of security, although it can still be used in secure networks or in controlled situations.

If it has not been installed, we can use (Debian) `apt-get install telnetd` and then verify that it has been registered either in /etc/inetd.conf or in /etc/xinetd.conf (or in the directory in the files are defined, for example, /etc/xinetd.d as indicated in the previous file with the sentence includes /etc/xinetd.d). xinetd.conf or /etc/xinetd.d/telnetd should include a section such as (any modification in xinetd.conf must reboot the service with `service xinetd restart`):

```
service telnet
{
disable = no
flags = REUSE
socket_type = stream
wait = nouser = root
server = /usr/sbin/in.telnetd
log_on_failure += USERID
}
```

Instead of using telnetd, we recommend using SSL telnet(d) which replaces telnet(d) using encryption and authentication through SSL or using SSH (next section). SSLTelnet(d) may work with telnet(d) normally in both directions, as, when beginning communication, it verifies whether the other peer supports SSL and if not, it continues with the normal telnet protocol. The advantages compared to telnet(d) are that the passwords and data do not pass through the network in the plain text mode and anyone using, for example, tcpdump will be able to see the contents of the communication. Also, SSLtelnet may be used to connect, for example, to a secure web server (for example https://servidor.web.org) by simply executing: `telnet server.web.org 443`.

3.2. Secure shell or SSH

An advisable change is to use ssh instead of telnet, rlogin or rsh. These latter commands are insecure (except for SSLTelnet) for various reasons: the most important is that all that is transmitted through the network, including the user names and passwords, is in plain text (although there are encrypted versions of telnet-telnetd, they must coincide in that both of them are encrypted), anyone that has access to that network or any segment of that network will be able to obtain all that information and then assume the identity of the user. The second is that these ports (telnet, rsh,...) are the first place at which a cracker will try to connect. The ssh protocol (in version OpenSSH) provides an encrypted and compressed connection that is much more secure than, for example, telnet (it is advisable to use version 2 of the protocol). All current distributions incorporate the ssh client and the sshd server by default.

3.2.1. ssh

To execute the command, proceed as follows:

ssh -l *login name host* o ssh *user@hostname*

Through SSH we can encapsulate other connections such as X11 or any other TCP/IP. If we omit the parameter -l, the user will connect to the same local user and in both cases the server will ask for the password to authenticate the user's identity. SSH supports different authentication modes (see ssh man pages) based on the RSA algorithm and the public password.

It is possible to create the user identification passwords using the command `ssh-keygen -t rsa|dsa`. The command creates in the user .shh directory the file id_rsa and id_rsa.pub, the private and public key respectively (for exemple, for RSA encryption algorithm). The user could copy the public key (id_rsa.pub) on the remote machine in the .ssh directory of the remote user, in the authorized_keys file. This file will be able to contain as many public keys as sites from which a remote connection to the machine will be wanted. The syntax is of one key per line and is equivalent to the .rhosts file (although the lines will have a considerable size). After entering the public keys of the user-machine into this file, this user will be able to connect from that machine without needing a password.

In normal mode (without creating the keys), the user will be prompted for a password, but the communication will always be encrypted and will never be accessible to other users who could be listening in on the network. For further information, see `man ssh`. In order to execute a command remotely, simply:

ssh -l *login name host_remote_command*

For example:

ssh -l user localhost ls -al

3.2.2. sshd

The sshd is the server (daemon) for ssh (if not installed, it can be installed using `apt-get install ssh` which will install the server and the client). In combination, they replace rlogin, telnet, and rsh and provide secure and encrypted communication between two insecure hosts in the network.

This will generally start up with the initialization files (/etc/init.d or /etc/rc) and wait for connections from clients. The sshd of most current distributions supports versions 1 and 2 of the SSH protocol. When the package is installed, it creates a specific RSA key of the host, and when the daemon boots, it creates another, the RSA for the session, which is not stored on disk and changes every hour. When a client initiates communication, the client generates a random number of 256 bits which is encrypted together with the two keys of the server and sent. This number will be used during the communication as the session key to encrypt the communication using a standard encryption algorithm. The user may select any of the available ones offered by the server. There are some (more secure) differences when using version 2 of the protocol. Then,

some of the user authentication methods described in the client are initiated or it will ask for the password, but always with the communication encrypted. For further information, see the sshd man pages.

3.2.3. Tunnel over SSH

Often we have access to an sshd server, but for security reasons not to other non-encrypted services (for example a POP3 mail service or X11 windows server) or simply we wish to connect to a service that can only be accessed from the company environment. To do so, it is possible to establish an encrypted tunnel between the client machine (for example with Windows, running a free software ssh client called putty) and the server with sshd. In this case, when we connect the tunnel to the service, the service will see the request as if it were coming from the same machine. For example, if we want to establish a POP3 connection on port 110 of the remote machine (which also has an sshd server) we will execute:

```
ssh -C -L 1100:localhost:110 user-id@host
```

This command will ask for the password of the user-id over the host and, once connected, the tunnel will have been created. Every package sent to the local machine over port 1100 will be sent to the remote machine localhost over port 110, which is where the POP3 service listens (option -C compresses the traffic through the tunnel).

Making tunnels over other ports is very easy. For example, let's suppose that we only have access to a remote proxy server from a remote machine (remote login) – not from the local machine –, we can make a tunnel to connect the navigator through the tunnel in the local machine. Let's suppose that we have a login on a `gateway` machine, which can access the machine called `proxy` that runs the Squid proxy server over port 3128. We run:

```
ssh -C -L 8080:proxy:3128 user@gateway
```

Once we have connected we will have the tunnel listening over local port 8080, which will redirect traffic from the gateway to the proxy to 3128. To navigate securely, all we will need to do is http://localhost:8080/

4. File transfer services: FTP

The file transfer protocol (FTP) is a client/server protocol (under TCP) which allows files to be transferred to and from a remote system. An FTP server is a computer that runs the ftpd daemon.

Some sites that allow an anonymous connection under anonymous user are generally software repositories. On a private site, we will need a username and password in order to obtain access. It is also possible to access an FTP server via a navigator and nowadays software repositories are usually replaced by web servers (e.g. Apache) or other technologies such as Bittorrent (which uses peer to peer (P2P) networks). Nonetheless, in some cases and with Debian, for example, access continues to use the username or password with the possibility of uploading files to the server (although this is also possible with web services). The file transfer protocol (FTP) (and servers/clients that implement it) by definition are not encrypted (the data, usernames and passwords are transmitted in clear text by the network) with its ensuing risk. But there are a number of servers/clients that support SSL and therefore, encryption.

4.1. FTP client (conventional)

An FTP client allows acces to FTP servers and there are a large number of clients available. Using FTP is extremely simple; from the command line, run:

ftp *server-name*

Or also FTP, and then interactively:

open *server-name*

The server will prompt for a username and a password (if it accepts anonymous users, anonymous will be entered as the username and our e-mail address as the password) and from the command prompt (following several messages) we will be able to start transferring files.

The protocol allows the transfer in ASCII or binary modes. It is important to decide what type of file has to be transferred because transferring a binary in ASCII mode will destroy the file. To change between modes, we will need to execute the ascii or binary command. Useful commands of the FTP client are the ls (navigation in the remote directory), get file_name (to download files) or mget (which admits *), put file_name (to send files to the server) or mput (which admits *); in these last two cases we need to be authorised to write on the server's directory. We can run local commands by entering a '!'

before the command. For example !cd /tmp will mean that the files downloaded to the local machine will be downloaded to /tmp. In order to view the status and functioning of the transfer, the client will be able to print marks, or ticks, which are activated by the hash and tick commands. There are other commands that can be consulted on the page of the manual (FTP man) or by running help from within the client.

We have numerous alternatives for clients, for example in text mode: ncftp, lukemftp, lftp, cftp, yafc, or in graphic mode: gFTP, WXftp, LLNL XFTP, guiftp. [Bor00]

4.2. FTP servers

The traditional UNIX server is run through port 21 and is booted by the inetd daemon (or xinetd depending on which one is installed). In inetd.conf it is advisable to include the tcpd wrapper with the access rules in host.allow and host.deny in the call to ftpd by inetd to increase the system's security (refer to the chapter on security). When it receives a connection, it verifies the user and password and allows entry if authentication is correct. An anonymous FTP works differently, since the user will only be able to access an established directory in the configuration file and its subjacent tree, but not upwards, for security reasons. This directory generally contains pub/, bin/, etc/, and lib/ directories so that the FTP daemon can run external commands for ls requests. The ftpd daemon supports the following files for its configuration:

- /etc/ftpusers: list of users that are not accepted on the system, one user per line.

- /etc/ftpchroot: list of users whose base chroot directory will be changed when they connect. Necessary when we want to configure an anonymous server.

- /etc/ftpwelcome: welcome announcement.

- /etc/motd: news after login.

- /etc/nologin: message shown after denying the connection.

- /var/log/ftpd: log of transfers.

If at some point we wish to inhibit the FTP connection, we can do so by including the /etc/nologin file. The ftpd will show its content and finish. If there is a .message file in a directory, the ftpd will show this when accessed.

A user's connection passes through five different levels:

1) Having a valid password.

2) Not appearing on the list /etc/ftpusers.

3) Having a valid standard shell.

4) If it appears in /etc/ftpchroot, it will be changed to the home directory (included if anonymous or FTP).

5) If the user is anonymous or FTP, it should have an entry in the /etc/passwd with FTP user, but will be able to connect by giving any password (conventionally the e-mail address is used).

It is important to bear in mind that the users that are only enabled to use the FTP service do not have a shell to the corresponding entry user in /etc/passwd to prevent this user having a connection through ssh or telnet, for example. Therefore, when the user is created, we will have to indicate, for example:

```
useradd -d/home/nteum -s /bin/false nteum
```

And then:

```
passwd nteum
```

Which will mean that the user nteum will not have a shell for an interactive connection (if the user already exists, we can edit the /etc/passwd file and change the last field for /bin/false). Then we will have to add as a last line /bin/false in /ect/shells. [Mou01] describes step by step how to create both a secure FTP server with registered users and an anonymous FTP server for non-registered users. Two of the most common non-standard servers are WUFT-PD (http://www.wuftpd.org) and ProFTPD (http://www.proftpd.org). [Bor00, Mou01]

To install Proftpd on Debian, execute: `apt-get install proftpd`. After it is downloaded, debconf will ask if we want to run it by inetd or in manual mode (it is advisable to select the latter). If we wish to stop the service (for example, in order to change the configuration), we can use `/etc/init.d/proftpd stop` and to modify the file we can use /etc/proftpd.conf.

Consult http://www.debian-administration.org/articles/228 in order to configure it in encrypted mode (TSL) or to have anonymous access.

A Debian server that is very interesting is PureFtpd (pure-ftpd) which is very secure, it allows virtual users, quotas, SSL/TSL, and a set of very interesting features. We can check its installation/configuration at http://www.debian-administration.org/articles/383.

5. Information exchange services at user level

5.1. The mail transport agent (MTA)

An mail transport agent (MTA) is responsible for sending/receiving mails from an e-mail server to/from Internet, implementing the simple mail transfer protocol (SMTP). By default, Debian uses exim, because it is easier to configure than other MTA packages, such as smail or sendmail (the latter is one of the precursors). exim offers advanced features such as rejecting known SPAM site connections, it has defences against junk mail or mail bombing and is extremely efficient at processing large amounts of mail. It is run through inetd on a line in the configuration file /etc/inetd.conf with parameters for normal configurations (or xinetd).

exim uses a configuration file in /etc/exim/exim.conf, which can be modified manually, but it is advisable to do so using a shell script called `eximconfig`, in order to be able to configure exim interactively. The configuration values will depend on the machine's situation; however, its connection is extremely easy, since the script itself suggests the default values. Nonetheless, in /usr/doc/exim we can find examples of typical configurations.

We can test whether the configuration is valid with exim-bV and, if there are errors in the configuration file, the program will show them on screen or, if everything is correct, it will simply indicate the version and date. To test if it can recognise a local mailbox, use:

```
exim -v -bt local_user
```

Which will show the layers of transport used and the user's local address. We can also do the following test with a remote user by replacing local user with a remote address to see how it behaves. Then try sending a local mail message and remotely, passing the messages directly to exim (without using an agent, for example, mailx), by keying in for example (all together):

```
exim postmaster@OurDomain
From: user@domain
To: postmaster@OurDomain
Subject: Test Exim
Test message
^D
```

Next, we can analyse the mainlog and paniclog track files in /var/log/exim/ to see its behaviour and see what error messages have been generated. Obviously, we can also connect to the system as the postmaster user (or as the user to which the mail has been sent) and read the mail messages to see if everything

is correct. The other way consists of running it in debug mode using -dNro as a parameter, where Nro is the debug level (1-9). The normal parameter with which we should boot it is `exim -bs`, whether by inetd or xinetd. It is also possible to run it as a daemon through `/etc/init.d/exim start` in systems that require a high mail processing capacity. See the documentation (included in Debian in the exim-doc-html package) in order to configure filters, verification of hosts, senders etc. It is also interesting to install the eximon package, which is an exim monitor that allows the administrator to see the queue of mail messages and logs and to act on messages in the queue in order to distribute them (freezing, bouncing, thawing...).

The latest version of exim is exim4 (it can be installed with `apt-get install exim4-daemon-heavy` (and also `install exim4-config` which will help to configure exim4) – bear in mind that there are different packages with different possibilities but exim4-daemon-heavy is the most complete. We recommend reading /usr/share/doc/exim/README.Debian.gz and update-exim4.conf(8). For further information, see the HowTo section http://www.exim.org/docs.html. One of the small differences to consider in the configuration is that instead of having a single configuration exim.conf (the default option if we install exim from the sources) the package exim4-config (it is advisable to install it) uses small configuration files instead of a single one and that these will be in /etc/exim4/conf.d/* and will be chained into a single file (/var/lib/exim4/config.autogenerated by default) by update-exim4.conf.

5.2. Internet message access protocol (IMAP)

This service allows access to mail messages stored in a single server through a mail client such as Thunderbird or the Seamonkey mail client (both in mozilla.org). This service supported by the imapd daemon (the current ones support the IMAP4rev1 protocol) allows an electronic mail file that is on a remote machine. The imapd service is offered through the 143 (*imap2*) or 993 (when SSL encryption is supported) (*imaps*) ports. If we use inetd, this server is booted through a line in /etc/inetd.conf as:

```
imap2 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
imap3 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
```

In this example, the tcpd wrapper is called, which functions with hosts.allow and hosts.deny in order to increase security. The most popular applications are uw-imapd (University of Washington and installed by default in Debian) or its secure version uw-imapd-ssl, but also cyrus-imap or courier-imap. To test that the imap server functions, we could use a client, such as `seamonkey -mail` and create an account for a local user and configure it appropriately so that it connects over the local machine, verifying that imap works correctly.

On Debian, the imap version has been compiled to support MD5 as the method for authenticating remote users, for encrypting connection passwords and to avoid replaced identities by sniffing on the network (the client used

to connect to the imap server must also support the MD5 authentication method). The method is very simple and secure, but the server must know the passwords in plain text of the mail users, meaning that it is advisable to use the version of imapd over SSL which functions over port 993. Like ssh, the imaps protocol is based on encrypting the communication through a host certificate (the client used for connecting to the server must also support this connection method, for example thunderbird or seamonkey -mail). To configure the imaps server, install the Debian package uw-imap-dsslwhich is the imap server with SSL support.

The installation will generate an auto-signed certificate valid for one year and stored in /etc/ssl/certs/imapd.pem. This certificate can be replaced by one signed by a certifying company or can generate its own one using OpenSSL. It is advisable to leave just the imaps entry in the file /etc/inetd.conf and to remove the imap2 and imap3 entries if we want the access to imap to be only by SSL.

Another protocol with similar characteristics which has been very popular in the past but that has been overtaken now by IMAP, is the post office protocol (POP) version 2 and 3. It is installed and booted in the same way as IMAP. There are numerous POP servers, but the most common ones are courier-pop, cyrus-pop3d, ipopd (University of Washington), qpopper, solid-pop3d.

5.2.1. Complementary aspects

Let's suppose that as users we have 4 email accounts on different servers and that we would like all email messages that are sent to these accounts to be gathered into a single one; to access that account externally and for it also to have an anti-spam filter.

First, we will have to install exim + Imap and check that they work. We need to take into account that if we install courier-imap (which according to some authors is better than uw-imapd) it functions over a mail format called Maildir, that exim will also have to be configured to run over maildir with the following configuration in /etc/exim/exim.conf (or the corresponding one if we have exim4), changing the option mail_dir format = true (the mails will be saved in the local user account in a directory called Maildir). Then we will have to reinitiate the exim server with `/etc/init.d/exim restart`, repeat the operational test by sending us an email message and read it with a client that supports maildir (for example `mutt -mailx` does not support it – see http://www.mutt.org).

To fetch the mail from the different accounts we will use fetchmail, (which is installed with `apt-get install fetchmail`). Next, we will have to create the .fetchmailrc file in our $HOME (we can also use the fetchmailconf tool) which will have to contain something like:

```
set postmaster "pirulo"
set bouncemail
set no spambounce
set flush

poll pop.domain.com proto pop3
user 'user1' there with password 'secret' is pirulo here

poll mail.domain2.com
user 'user5' there with password 'secret2' is 'pirulo' here
user 'user7' there with password 'secret3' is 'pirulo' here
```

The action set tells Fetchmail that this line contains a global option (error sending, delete mail from servers...). Next, we will specify the mail servers: one for checking if there is mail with the POP3 protocol and another for testing the use of several protocols to find one that works. We check the mail of two users with the second server option, but all mail found is sent to pirulo's mail spool. This allows us to check several mailboxes of different servers as if they were a single MUA mailbox. The specific information of each user starts with the action user. The fetchmail can be put in the cron (for example in /var/spool/cron/crontabs/pirulo adding 1 * * * * /usr/bin/fetchmail -s), so that it runs automatically or can be run in daemon mode (put set daemon 60 in .fetchmailrc and run it once for example in Autostart of Gnome/KDE or in .bashrc – it will run every 60 seconds).

To remove junk mail we will use SpamAssassin (apt-get install spamassassin) and we can configure Kmail or Evolution (check the bibliography to see how to configure it) for them to run it. In this configuration we will use Procmail, which is a very powerful tool (it allows mail distribution, filtering, automatic resending...). Once installed (apt-get install procmail), we need to create a file called .procmailrc in each user's home which will call the Spamassassin:

- Set *yes* for functioning or debugging messages
  VERBOSE=no

- We suppose that the mails are in "~/.Maildir", change if it is another
  PATH=/usr/bin:/bin:/usr/local/bin:
  MAILDIR=$HOME/Maildir
  DEFAULT=$MAILDIR/
  # Directory for storing the files
  PMDIR=$HOME/.procmail
  # Comment if we do not want a log of Procmail
  LOGFILE=$PMDIR/log
  # Spam filter
  INCLUDERC=$PMDIR/spam.rc

The file ~/.procmail/spam.rc contains:

If the spamassassin is not on the PATH
add the directory to the PATH variable:

```
# 0fw: spamassassin.lock|
| spamassassin - a

# The three following lines will move
# Spam mail to a directory called
# "spam-folder". If we want to save it in the Inbox, so that
# it can be filtered later with the client, comment the three lines.

:0:
* ^X-Spam-Status: Yes
spam-folder
```

The file ~/.spamassassin/user_prefs contains some useful configurations for spamassassin (see the bibliography):

```
#User preferences file. Ver man
#Mail::SpamAssassin::Conf
#Threshold for recognising a Spam: #Default 5, but with 4 it works a bit better
required_hits 4
# Sites we will never consider Spam to
#come from
whitelist_from root @debian.org
whitelist_from *@uoc.edu
#Sites SPAM always comes from
#(separated by commas)
blacklist_from viagra@domain.com
#Addresses on Whitelist and blacklist are
#global patterns such
#as:"friend@place.com", "*@isp.net", or
#"*.domain.com".
#Insert the word "[SPAM]" in the subject
#(to make filtering easier).
#If we do not wish to comment the line.
subject_tag [SPAM]
```

This will generate a X-Spam-Status tag: *Yes* in the message heading if it believes that the message is Spam. Then we will have to filter these and put them in another file or to delete them directly. We can use procmail to filter mails from domains, users etc. For further information, visit http://www.debian-administration.org/articles/242. Finally, we can install a mail client and configure filters so that it selects all email messages with X-Spam-Status: *Yes* and deletes them or sends them to a directory where we will later verify false positives (mails identified as junk but that are not). A complementary aspect of this installation is if we wish to have a mail server through webmail (in other words, to be able to check the mails from a server through a navigator without having to install or configure a client – like consulting a gmail or hot-

mail account) it is possible to install Squirrelmail (`apt-get install squirrelmail`) in order to offer this service. For Debian visit http://www.debian-administration.org/articles/200.

There are other possibilities as discussed at http://www.debian-administration.org/articles/364 installing MailDrop instead of Procmail, Postfix instead of Exim, or including Clamav/Amavisd as an antivirus (Amavisd allows postfix to be linked with spamassassin and clamav).

5.3. News

The news or discussion groups are supported through the Network News Transfer Protocol (NNTP). Installing a news server is necessary if we wish to read news offline, if we wish to have a repeater of the central servers or if we wish to have our own news master server. The most common servers are INN or CNEWS, but they are complex packages designed for large servers. Leafnode is a USENET package that implements a TNP server, especially suited for sites with small groups of users but from which we wish to access a large number of news groups. This server is installed in the basic Debian configuration and can be reconfigured with `dpkg-reconfigure leafnode` for all parameters such as central servers, type of connection etc. This daemon starts up from inetd in a similar way as imap (or with xinetd). Leafnode supports filters through regular indicated expressions (of the type ^Newsgroups:. * [,] alt.flame$) in /etc/news/leafnode/filters, where for each message the heading is compared to the regular expression and if there is a match, the message is rejected.

This server is simple to configure and all the files must be the property of a news user with authorisation to write (check that this owner exists in /etc/passwd). All control, news and configuration files are found in /var/spool/news except for the configuration of the server itself which is in the /etc/news/leafnode/config file. The configuration has some obligatory parameters that must be configured (for example, so that the server can connect to the master servers). They are *server* (news server from which the news will be obtained and sent) and *expire* (number of days that a thread or session has been read and will be deleted). Likewise, we have a set of optional parameters of a general or specific nature to the server that can be configured. For further information, see the documentation (*leafnode man* or /usr/doc/leafnode/README.Debian).

To check the server performance, we can run:

telnet localhost nntp

and if everything works correctly, it will show the server identification and will wait for a command, as a test, we can enter *help* [to abort, Ctrl+ (and then Quit)].

5.4. World Wide Web (httpd)

Apache is one of the most popular servers with the best capabilities in terms of hypertext transfer protocol (HTTP). Apache has a modular design and supports dynamic module extensions during its execution. It is highly configurable in the number of servers and available modules and supports various mechanisms of authentication, access control, metafiles, proxy caching, virtual servers etc. With modules (included in Debian) it is possible to have PHP3, Perl, Java Servlets, SSL and other extensions (see the documentation in http://www.apache.org).

Apache is designed to be executed as a daemon standalone process. This way it creates a set of subsidiary processes that will handle entry requests. It can also be executed as an Internet daemon through *inetd*, meaning that it will start up every time it receives a request. The server's configuration can be extremely complex depending on the requirements (check the documentation), however, here we can see a minimum acceptable configuration. The configuration files are in /etc/apache and are httpd.conf (main configuration file), srm.conf, access.conf (these last two are maintained for compatibility), mime.conf (MIME formats) and magic (file identification number). The log files are in /var/log/apache and are error.log (registers the errors in the server requests), access.log (register of who has accessed what) and apache.pid (process identifier).

Apache boots from the start up script /etc/init.d/apache and /etc/rcX.d, but can be controlled manually through the apachectl command. The apacheconfig command can also be used in order to configure the server. The default directories (in Debian) are:

- /var/www: directory of HTML documents.
- /usr/lib/cgibin: directory of executables (*cgi*) by the server.
- http://server.domain/ user: users' personal pages.
- /home/~user/public.html: directory of personal pages.

The default file that is read from each directory is index.html. After installing the *apache* and *apache-common* packages, Debian basically configures the server and initiates it. We can check that it functions by opening a browser (for example, the Konqueror, and typing "http://localhost" in the URL bar, which will load the page /var/www/index.html).

5.4.1. Manual (minimum) configuration of httpd.conf

Let's look at some of the most important parameters to be configured in Apache (the example is taken from Apache version 1.X and there are some minor changes if we use version 2).

ServerType standalone	Recommended, more efficient
ServerRoot /etc/apache	Where the configuration files are found
Port 80	Where the server will listen to requests
User www-data	User and group with which the server will be executed (important for security) must be valid users (they can be *locked*)
Group www-data ServerAdmin webmaster@pirulo.remix.com	User address that will attend to errors
ServerName pirulo.remix.com	Name of the server sent to users – must be a valid name in /etc/host or DNS –
DocumentRoot /var/www	Directory where the documents will be
Alias /icons/ /usr/share/apache/icons/	Where the icons are
ScriptAlias /cgibin/ /usr/lib/cgibin/	Where the CGI scripts are

5.4.2. Apache 2.2 + SSL + PHP + MySQL

An important aspect of dynamic web servers is making the most of the advantages of Apache in secure mode (SSL), PHP (is programming language generally used to create web site content) and MySQL+PHPAdmin (database that will be discussed in later chapters and graphic interface for managing it) all working in combination. We will start by installing it on a Debian Sarge, but not through the deb packages but rather from the software downloaded from the relevant sites, this way we can repeat the experience with other distributions. Obviously, afterwards it will not be possible to control these packages using apt or another package manager. We need to take care with the versions, which can change, and not to install the package over already installed packages.

a) Download the necessary files (for example within the directory /root -> cd /root):

1) Apache: from http://httpd.apache.org/download.cgi: httpd-2.2.4.tar.bz2

2) PHP: from http://www.php.net/downloads.php PHP 5.2.1 (tar.bz2)

3) MySQL from http://mysql.org/get/Downloads/MySQL-4.1/mysql-standard-4.1.21-pc-linux-gnu-i686.tar.gz/from/pick

4) PHPAdmin from http://prdownloads.sourceforge.net/phpmyadmin/phpMyAdmin-2.9.1-all-languages.tar.bz2?download

b) Utilities: bzip2 libssl-dev openssl gcc g++ cpp make (verify that they are not installed or otherwise, run apt-get install bzip2 libssl-dev openssl gcc g++ cpp make.

c) Apache:

```
cd /root
tar jxvf httpd-2.2.4.tar.bz2
cd httpd-2.2.4
```

With prefix, we indicate that we will install for example `/usr/local/apache2`

```
./configure --prefix=/usr/local/apache2 \
–with ssl=/usr/include/openssl \
--enable-ssl
make
make install
```

We modify the configuration file /usr/local/apache2/conf/httpd.conf andchange the user and workgroup for www-data:

User www-data
Group www-data

We change the owner and group of the data directory to

www-data:chown -R www-data:www-data /usr/local/apache2/htdocs

We modify the user www-data to change its home directory in /etc/passwd:

www-data:x:33:33:www-data:/usr/local/apache2/htdocs:/bin/sh

Apache server installed. To initiate it (to stop it, change *start* for *stop*):

/usr/local/apache2/bin/apachectl start

We can place a script to start up the apache server upon booting.

ln -s /usr/local/apache2/bin/apachectl /etc/rcS.d/S99apache chmod 755 /etc/rcS.d/S99apache

d) SSL:

In /usr/local/apache2/conf/httpd.conf we remove the comment from the line
Include conf/extra/httpd-ssl.conf

The files are generated with the keys for the secure server, in /root we run (adjust the versions to the ones that have been downloaded) – the first openssl command is a long line and ends with 1024:

```
openssl genrsa -rand ../httpd-2.2.4.tar.bz2:../php-5.2.1.tar.bz2:../phpMyAdmin-2.9.1-
all-languages.tar.bz2 -out server.key 1024
openssl rsa -in server.key -out server.pem
openssl req -new -key server.key -out server.csr
openssl x509 -req -days 720 -in server.csr -signkey server.key -out server.crt
```

We copy the files...

```
cp server.crt /usr/local/apache2/conf/
cp server.key /usr/local/apache2/conf/
```

We restart the server...

```
/usr/local/apache2/bin/apachectl restart
```

We can check how to add the SSL module to a server that does not have it installed at http://www.debian-administration.org/articles/349.

e) MySQL (for more information see module 8):

We create a group and a user for MySQL if it does not exist.

```
groupadd mysql
useradd -g mysql mysql
```

In the directory where we will install MySQL (/usr/local/) we type:

```
cd /usr/local/
gunzip < /root/mysql-standard-4.1.21-pc-linux-gnu-
i686.tar.gz | tar xvf - ln -s mysql-standard-4.1.21-pc-
linux-gnu-i686 mysql cd mysql
```

We create a database and change the permissions

```
scripts/mysql_install_db --user=mysql
chown -R root.
chown -R mysql data
chgrp -R mysql.
```

We can place a script for initiating the mySQL server.

```
ln -s /usr/local/mysql/support-files/mysql.server /etc/
rcS.d/S99mysql.server
chmod 755 /etc/rcS.d/S99mysql.server
```

We start the server

```
/etc/rcS.d/S99mysql.server start
```

We can enter the database and change the password of the root user for security (consult http://dev.mysql.com/doc/refman/5.0/en/index.html for the syntax)
`/usr/local/mysql/bin/mysql`

Inside, we can type:
USE mysql

We place the password pirulo on the user root
UPDATE user SET Password=PASSWORD('pirulo') WHERE User='root';
FLUSH privileges;

To enter MySQL we will have to type
`/usr/local/mysql/bin/mysql -u root -ppirulo`

f) PHP (replace with the appropriate versions):

Necessary utilities:

```
apt-get install libxml2-dev curl \
libcurl3-dev libjpeg-mmx-dev zlib1g-dev \
libpng12-dev
```

With the Apache server stopped we can type:

```
cd /root
tar jxvf php-5.2.0.tar.bz2
cd php-5.2.0
```

With the prefix we can indicate where we want to install it (all on one line):

```
./configure  --prefix=/usr/local/php5  --enable-mbstring
--with-apxs2=/usr/local/apache2/bin/apxs  --with-mysql=/
usr/local/mysql  --with-curl=/usr/include/curl  --with-
jpeg-dir=/usr/include  --with-zlib-dir=/usr/include  --
with-gd --with-xml --enable-ftp --enable-bcmath

make
make
install cp php.ini-dist /usr/local/php5/lib/php.ini
```

We modify Apache (/usr/local/apache2/conf/httpd.conf) in the indicated part:

```
<IfModule mime_module>
    AddType application/x-httpd-php .php .phtml
    AddType application/x-httpd-php-source .phps
```

And also:

DirectoryIndex index.php index.html

We restart the server.

g) PHPAdmin

```
cd /usr/local/apache2/
```

The phpmyadmin is decompressed in the apache2 directory (be careful with the versions).

```
tar jxvf /root/phpMyAdmin-2.9.1-all-languages.tar.bz2
mv phpMyAdmin-2.9.1-all-languages phpmyadmin
cd phpmyadmin
cp config.sample.inc.php config.inc.php
```

We need to modify the configuration file (config.inc.php):

```
$cfg['blowfish_secret'] = 'pirulo';
```

We remove the user and user password by default two quotation marks (') one after the other:

```
$cfg['Servers'][$i]['controluser'] = '';
$cfg['Servers'][$i]['controlpass'] = '';
```

We change apache (/usr/local/apache2/conf/httpd.conf) adding in <IfModule alias_module>

```
<IfModule alias_module>
    Alias /phpmyadmin "/usr/local/apache2/phpmyadmin/"
<Directory "/usr/local/apache2/phpmyadmin/">
    Order allow, deny
    Allow from all
</Directory>
```

We reinitiate the server and we can it call with http://localhost/phpadmin

Further information can be obtained from the respective websites of each application and in LWP.

6. Proxy Service: Squid

A Proxy server (PS) is used to save connection bandwidth, to improve security and to increase web-surfing speed.

Squid is one of the main PS, since it is OpenSource, it accepts ICP (characteristics that allow the exchange of hints with other PS), SSL (for secure connections between proxies) and supports FTP objects, Gopher, HTTP and HTTPS (secure). Its functioning is simple, it stores the most frequently requested objects in the RAM and the least requested objects in a database on the disk. Squid servers can also be configured hierarchically to form a tree of proxies according to requirements. There are two possible configurations:

1) As an httpd accelerator to achieve improved performance of the web service.

2) As a proxy-caching server to allow the users of a corporation to use the PS to exit towards the Internet.

In the first mode, it acts as an inverse proxy in other words, it accepts a client's request, serves the object if it has it, and if not, asks for it and passes it onto the client when it does, storing it for the next time. In the second option it can be used as a control to restrict the sites where a connection to the Internet can be obtained or to authorise access at specific times of day. Once installed (squid package in Debian, squid-cgi, squidguard or squidtaild can also be installed) three files are generated: /etc/squid.conf (configuration), /etc/init.d/squid (initialisation) and /etc/logrotate.d/squid (for log control).

6.1. Squid as an http accelerator

In this mode, if the web server is on the same machine as the PS, it will have to be reconfigured to attend to the requests of port 81 (in Apache, change Port 80 for Port 81 in httpd.conf). The configuration file (/etc/squid.conf) contains a large number of entries, but here we will only see the essential ones [Mou01]:

http_port 80 icp_port 0 hierarchy_stoplist cgi-bin \? acl QUERY urlpath_regex cgi-bin \? no_cache deny QUERY	Where it listens for httpd Where it listens for ICP
cache_mem 100 MB redirect_rewrites_host_header off cache_replacement_policy lru memory_replacement_policy lru	Memory for objects in progress

cache_dir ufs /var/spool/squid 100 16 256 Database emulate_httpd_log on	Type and place where we can find the disk cache
acl all src 0.0.0.0/0.0.0.0 http_access allow all cache_mgr root cache_effective_user proxy cache_effective_group proxy httpd_accel_host192.168.1.1 httpd_accel_port 81 logfile_rotate 0 log_icp_queries off buffered_logs on	Access for all And for everything Mail responsible UID GID Real web server Port

In this way, the option httpd_accel_host deactivates the possibility of it being executed as proxy-caching. For further information visit http://www.squid-cache.org/.

6.2. Squid as proxy-caching

This way, squid is enabled to control Internet access, when access will be given, the object that can be accessed. In this case, the configuration file will have to include the following modifications added in /etc/squid.conf:

```
acl localnet src 192.168.1.0/255.255.255.0
acl localhost src 127.0.0.1/255.255.255.255
acl Safe_ports port 80 443 210 70 21 102565535
acl CONNECT method CONNECT
acl all src 0.0.0.0/0.0.0.0
http_access allow localnet
http_access allow localhost
http_access deny
http_access deny CONNECT
http_access deny all
cache_emulate_httpd_log on
```

The main difference with the other mode are the acl lines, in which case C class clients C 192.168.1.0 will be allowed access to the PS, also the localhost IP and other ports that will be able to access the Internet 80(http), 443(https), 210(whais), 70(gopher), and 21(ftp), also, the *connect* method is denied to avoid a connection from the outside to the PS and then all IP and ports over the PS are denied. [Mou01] More information at http://www.squid-cache.org/ and for a transparent-proxy at http://tldp.org/HOWTO/TransparentProxy-1.html.

7. OpenLdap (Ldap)

LDAP means lightweight directory access protocol and is a protocol for accessing data based on an X.500 service. It is run on TCP/IP and the directory is similar to that of a database that contains information based on attributes. The system allows this information to be organised in a secure manner and uses replicas to maintain its availability, ensuring its coherence and verification of accessed-modified data.

The service is based on the client-server model, where there is one or more servers that contain the data; when a client connects and asks for information, the server replies with the data or a pointer to another server where more information can be extracted, but the client will only see a directory of global information. [Mou01, Mal07]

To import and export information between ldap servers or to describe a number of changes that will be applied to the directory, we use a format called LDIF (LDAP *data interchange format*). LDIF stores the information in hierarchies oriented at objects that will then be converted to the internal format of the database. An LDIF file has a similar format to:

```
dn: o = UOC, c = SP
or: UOC
objectclass: organization
dn: cn = Pirulo Nteum, o = UOC, c = SP
cn: Pirulo Nteum
sn: Nteum
mail: nteum@uoc.edu
objectclass: person
```

Each entry is identified by a name indicated as a distinguished name (DN). The DN consists of the entry name plus a series of names that relate it to the directory's hierarchy and where there is an *objectclass*, that defines the attributes that can be used in this entry. LDAP offers a basic set of object classes: groups (including disorganised lists of individual objects or groups of objects), locations (such as countries and their description), organisations and people. An entry can also belong to more than one object class, for example, an individual is defined by the class of person, but can also be defined by attributes of the classes inetOrgPerson, groupOfNames, and organisation. The structure of the server's objects (called *schema*) determines what the permitted attributes are for an object of a class (which are defined in /etc/ldap/schema as opeldap.schema, corba.schema, nis.schema, inetorgperson. schema etc.).

All the data are represented as a pair *attribute* = *value* where attribute is the description of the information it contains, for example, the attribute used to store the name of a person is commonName, or cn, in other words for a person called Pirulo Nteum, it will be represented by cn: Pirulo Nteum and will have associated other attributes of the person class such as givenname: Pirulo surname: Nteum mail: pirulo@uoc.edu. The classes have obligatory and optional attributes and every attribute has an associated syntax which indicates what type of information the attribute contains, for example, bin (*binary*), ces (*case exact string*, case must match in search), cis (*case ignore string*, case can be ignored during the search), tel (*telephone number string*, ignores spaces and '-'), dn (*distinguished name*). An example of a file in LDIF format could be:

```
dn: dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
dn: ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
ou: groups
dn: ou = people, dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
ou: people
dn: cn = Pirulo Nteum, ou = people, dc = UOC, dc = com
cn: Pirulo Nteum
sn: Nteum
objectclass: top
objectclass: person
objectclass: posixAccount
objectclass: shadowAccount
        uid:pirulo
        userpassword:{crypt}p1pss2ii(0pgbs*do&@ = )eksd
        uidnumber:104
        gidnumber:100
        gecos:Pirulo Nteum
        loginShell:/bin/bash
        homeDirectory: /home/pirulo
        shadowLastChange:10877
        shadowMin: 0
        shadowMax: 999999
        shadowWarning: 7
        shadowInactive: -1
        shadowExpire: -1
        shadowFlag: 0
dn: cn = unixgroup, ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: posixGroup
cn: unixgroup
gidnumber: 200
memberuid: pirulo other-user
memberuid:
```

The long lines can be continued underneath starting with a space or a tab (LDIF format). In this case, the DN base has been defined for the institution dc = UOC, dc = com, which contains two sub-units: *people* and *groups*. Then it has described a user that belongs to *people* and to *group*. Having prepared the file with the data, we need to import it to the server so that it is available for LDAP clients. There are tools for converting the data of different databases to the LDIF format. [Mal07]

In Debian, we need to install the slapd package which is the OpenLdap server. During the installation, it will ask a number of questions such as: *Method of installing the directory*: auto; *extensions to the directory [domain-host,site,institution]: host, domain, password of the Adm; replicate local changes to other servers*: no. This installation will generate a configuration file in /etc/ldap/slapd.conf and the database on /var/lib/ldap. There is also another file /etc/ldap/ldap.conf (or ~/.ldaprc), which is the configuration file used for initialising default values when ldap clients are executed. Here it indicates which is the database, which is the ldap server, security parameters, size of the search etc.

The /etc/ldap/slapd.conf server configuration file (see `man slap.con`) consists of different sections, each indicated by one of the following guidelines: global, backend specific and database specific, and in that order. The *global* guideline is of a general nature and applies to all the *backends* (databases) and defines general questions such as access permissions, attributes, waiting times, *schemas* etc. The *backend specific* guideline defines the attributes to the specific *backend* that it defines (bdb, dnssrv, ldbm...), and the *database specific* guideline defines the specific attributes for the database it defines. To boot the server, we need to run:

```
/etc/init.d/slapd start (or stop to stop it)
```

During the installation, the system will have created the right links for running it after start up.

7.1. Creating and maintaining the database

There are two methods for entering data in the LDAP database. The first one is easy and suitable for small amounts of data, it is interactive and we need to use tools such as ldapadd (or any other such as Ldap Browser http://www.iit.edu/~gawojar/ldap/) to make new entries. The second needs to be worked on offline and is suitable for large databases and uses the slapadd command included with slapd. Because it is more general, we will briefly describe the second method, where we must first verify that it contains the following attributes in slapd.conf: suffix (top of the directory, for example "o = UOC, c = SP"); directory /var/lib/ldap (directory where the indexes will be created and that can write slapd). We must also verify that the database contains the definitions of the indexes we wish to use:

```
index cn,sn,uid
index objectClass pres,eq
```

Having defined the slapd.conf, we must execute the command:

```
slapadd -l entry-f configuration [-d level] [-n whole| -b
suffix]
```

The arguments are:

-*l:* file in LDFI format.
-*f*: server configuration file where it indicates how to create the indexes.
-*d*: level of debugging.
-*n*: Nro of the database, if we have more than one.
-*b*: specifies what database need to be modified.

There are other commands with slapd such as slapindex, which allows the indexes to be regenerated, and slapcat, which allows dumping the database to a file in LDIF format.

8. File services (NFS)

The NFS system allows a server to export a file system so that it can be used interactively from a client. The service consists of an nfsd server and a client (*mountd*) which can share a file system (or part of it) through the network.

In Debian, install `apt-get install nfs-common portmap` for the client, while the server needs:
`apt-get install nfs-kernel-server nfs-common portmap`.

The server (in Debian) starts through the nfscommon and nfs-kernel-server scripts in /etc/init.d (and the appropriate links in /etc/rcX.d).

The server uses a file (/etc/exports) to manage the access and control of the file systems that will be accessed remotely. On the client, the root (or other user through *sudo*) can mount the remote system using the command:

```
mount IPserver:remote-directory local_directory
```

and as of that moment, the remote-directory will be seen within the local directory (which must exist before executing the *mount*). This task in the client can be automated using the automatic *mount* file (/etc/fstab) including a line; for example:

pirulo.remix.com:/usr/local /pub nfs rsize=8192,wzise=8192,timeo=14

This sentence indicates that the directory /usr/local of the host pirulo.remix.com will be mounted in the /pub local directory. The parameters rsize, wzise are the size of the reading and writing blocks, timeo is the RPC timeout (if these three values are not specified, the default values are taken).

The /etc/exports file serves as ACL (access control list) of the file systems that can be exported to the clients. Every line contains a file system to be exported followed by the clients that can mount it, separated by blank spaces. Each client can have a set of options associated to it in order to modify the behaviour (see the *exports man* for a detailed list of the options). An example of this could be:

```
# Example of /etc/exports
/ /master(rw) trusty(rw,no_root_squash)
/projects proj*.local.domain(rw)
/usr *.local.domain(ro) @trusted(rw)
/pub (ro,insecure,all_squash)
/home 195.12.32.2(rw,no_root_squash) www.first.com(ro)
```

/user 195.12.32.2/24(ro,insecure)

The first line exports the entire file system (/) to master and trusty in read/write mode. Plus, for trusty there is no *uid squashing* (the root of the client will access as root the root files of the server, in other words, the two root are equivalent despite being from different machines; it is suited for machines without a disk). The second and third lines show examples of '*' and *netgroups* (indicated by @). The fourth line exports the */pub* directory to any machine in the world, read-only, allows access to NFS clients that do not use a port reserved for NFS (option *insecure*) and everything is executed under the user *nobody* (option *all squash*). The fifth line specifies one client for its IP and the sixth the same but with a network mask (/24) and with options between brackets () and without any spaces. There can only be spaces between the enabled clients. It is important to bear in mind that there are 3 versions of NFS (V2, V3 and recently V4). The most common ones are V3 and in some installations V2. If from a V3 client we connect to a V2 server, this situation must be indicated with a parameter.

8.1. Wiki server

A wiki (from Hawaiian *wiki wiki*, "fast") is a collaborative website that can be edited by various users who can create, edit, delete or modify the content of a web page, in an easy, fast and interactive manner; these capabilities make wiki an effective tool for collaborative writing. Wiki technology allows web pages stored in a public server (the wiki pages) to be written in a collaborative fashion through a navigator, using simple notation for giving format, creating links etc., saving a log of changes that makes it possible to recover easily any prior status of the page. When someone edits a wiki page, its changes appear immediately on the web, without passing through any type of prior revision. Wiki can also refer to pages of hypertext, which can be visited and edited by anyone (definition of Wikipedia). Debian has its wiki in http://wiki.debian.org/ and FC in http://fedoraproject.org/wiki/ and both are based on Moin Moin (http://moinmoin.wikiwikiweb.de/). MoinMoin is a Python WikiClone that can rapidly initiate its own wiki; it just needs a web server and the installed Python language.

In http://moinmoin.wikiwikiweb.de/MoinMoinPackages/DebianLinux we can find detailed instructions for installing Moin Moin on Debian, but, basically, it comes down to: 1) Installing apache2 and mod_python, 2) configuring Apache to note the code of MoinMoin, 3) installing the moinmoin package, 4) configuring MoinMoin and 5) restarting Apache. A configuration example:

```
apt-get install python-moinmoin
mkdir /var/www/mywiki
cp -r /usr/share/moin/data /usr/share/moin/underlay \
/usr/share/moin/server/moin.cgi /var/www/mywiki
chown -R www-data:www-data /var/www/mywiki
```

```
chmod -R g+w /var/www/mywiki
```

- Configure apache2 by adding /etc/apache2/conf.d/wiki (or wherever the configuration file is):
  Alias /wiki/ "/usr/share/moin/htdocs/"
  <Location /mywiki>
  SetHandler python-program
  PythonPath "['/var/www/mywiki','/etc/moin/']+sys.path"
  PythonHandler MoinMoin.request::RequestModPy.run
  PythonDebug On
  </Location>

- Restart apache2:
  `/etc/init.d/apache2 reload`

- Configure MoinMoin: Edit /etc/moin/farmconfig.py (multiple wikis)
  wikis = [
  ("mywiki", r"^yoursite.com/mywiki/.*$"),
  ]

- we can also use (just one wiki):
  wikis = [
  ("mywiki", r".*"),
  ]

- Also in /etc/moin/farmconfig.py remove the comment data_dir and data_underlay_dir (one for each wiki) and copy the file.
  `cp /etc/moin/moinmaster.py /etc/moin/mywiki.py`

- Then edit /etc/moin/mywiki.py and change:
  sitename = u'MyWiki'
  data_dir = '/var/www/mywiki/data'
  data_underlay_dir = '/var/www/mywiki/underlay'

The Wiki will be installed on http://yoursite.com/mywiki/

Activities

1) Configure a DNS server as cache and with its own domain.

2) Configure a NIS server/client with two machines exporting the server's user directories by NFS.

3) Configure an SSH server to access from another machine without a password.

4) Configure an Apache server + SSL+ PHP+ MySQL+ PHPAdmin in order to visualise users' personal pages.

5) Create and configure an electronic mail system through Exim, fetchmail, Spam-Assassin and an IMAP server for receiving mail from the outside and being able to read them from a remote machine with the Mozilla client (Thunderbird).

6) Install the MoinMoin Wiki and create a set of pages to verify that it works.

Bibliography

Other sources of reference and information

[Debc, LPD03b, Ibi]

http://tldp.org/HOWTO/DNS-HOWTO-7.html

http://tldp.org/HOWTO/NIS-HOWTO/verification.html

Squid proxy server

Proxy Cache: http://www.squid-cache.org/

Transparent Proxy: http://tldp.org/HOWTO/TransparentProxy-1.html

Proftpd: http://www.debian-administration.org/articles/228

PureFtpd: http://www.debian-administration.org/articles/383

Exim: http://www.exim.org/docs.html

Mutt: http://www.mutt.org

ProcMail: http://www.debian-administration.org/articles/242

LWP:http://www.lawebdelprogramador.com/temas/tema_stablephpapachemysql.php

Moin Moin: (http://moinmoin.wikiwikiweb.de/)

Moin Moin + Debian:

http://moinmoin.wikiwikiweb.de/MoinMoinPackages/DebianLinux

Apache2 + SSL: http://www.debian-administration.org/articles/349