Each user can now create in his personal folder a *public_html* directory where to place his pages. How to use the personal folder is explained in more detail here.

This option is not enabled for the *root* user.

This type of configuration has been used a lot for example in universities where every professor had a web page available just by having a user. You cannot test the configuration of this directive on your server yet, but you can read the documentation of the directive Which users now have a personal folder? The default path to the personal folder is not very easy to type with a keyboard in many countries, how would you change it? Considering a remote server,
What disadvantages do you see to this way of working? If you had to propose a similar functionality t o d a y ,
how would you do it?

In the module documentation you can see that the directory in which each user can create their pages is highly configurable.

Activate the mod_userdir module and create a folder for the user with which you installed Linux. Inside the folder create a test.html file and configure the module so that when accessing the user's personal folder from a web browser, this file is displayed by default.

Finally, deactivate the module.

The MIME module

MIME types are used to identify file types. This module is used for that and some other things. There is a ready-made configuration in *mime.conf*.

TypesConfig

Indicates where the file describing the MIME types (mime.types) will be, or its equivalent if we have changed it. There are not many reasons for this, so we should leave the default value.

```
TypesConfig conf/mime.types
```

DefaultType

Set the MIME type for all those files that cannot be assigned a MIME type by extension, etc. If we have a server where most of the content is HTML pages, XML, etc. it is a good idea to use the default value, but if most of the content is binary files (pictures, programs, etc.) it would be convenient to set *application/octet-stream*.

```
DefaultType text/plain
```

Further specification is possible:

```
DefaultType text/html
```

Research on the Internet what MIME types are and which ones are there. What body is in charge of managing them? What other things is it in charge of?

AddEncoding

Specifies a particular type of encoding for certain file extensions. *AddEncoding* can also be used to tell browsers to decompress certain files while downloading them:

```
AddEncoding x-compress Z
AddEncoding x-gzip gz
```

AddLanguage

It is used to associate extensions to specific languages for content. This directive is useful for content negotiation between the server and the client's web browser since Apache can return content in different languages depending on the language setting of the web browser. It is useful especially in cases where we have the site written in multiple languages as it will allow Apache to serve the appropriate one in each case depending on the configuration of the client's browser. This is done transparently to the end user.

The language codes are defined in the ISO 639 specification. More specifically in ISO 639-1 for two-letter codes and in ISO 639-2 for three-letter codes that include more languages.

```
# Danish (da) - Dutch (nl) - English (en) - Estonian (et)
# French (fr) - German (de) - Modern Greek (el)
# Italian (it) - Norwegian (no) - Korean (kr)
# Portuguese (pt) - Luxembourgish(ltz)
# Spanish (es) - Swedish (sv) - Czech(cz)
# Polish (pl) - Brazilian Portuguese (pt-br) - Japanese
(ja) # Russian (ru) - Croatian (hr)
#
AddLanguage es .es
AddLanguage da .dk
AddLanguage nl .nl
AddLanguage en .en
AddLanguage et .et
AddLanguage fr .fr
AddLanguage de .de
AddLanguage el .el
AddLanguage it .it
AddLanguage ja .ja
AddLanguage pl .po
AddLanguage kr .kr
AddLanguage pt .pt
AddLanguage no .no
AddLanguage pt-br .pt-br
AddLanguage ltz .ltz
AddLanguage sv .se
AddLanguage cz .cz
AddLanguage ru .ru
AddLanguage hr .ru
AddLanguage tw .tw
AddLanguage hr .hr
```

Note that the language indicator does not have to be identical to the suffix as seen in several of the examples.

AddCharset

It is the same as *AddLanguage* but for adding new character sets.

```
AddCharset ISO-8859-1 .iso8859-1 .latin1
AddCharset ISO-8859-2 .iso8859-2 .latin2 .cen
```

Virtual Hosts

Apache can serve multiple web sites from a single web server. The client will never differentiate whether they are sites on different servers or on the same machine. Apache is a very powerful server for using this option.

If for example we are the managers of two DNS domains (www.laempresa.es and www.mipagina.es) we can host both sites on the same Apache server. One of these domains will be considered the main site and all the others will be virtual hosts.

Advantages:

1. Leverage existing hardware.
2. Take advantage of available public IP addresses.

A great advantage in the use of virtual hosts in Apache is that it allows to inherit the configuration of the main site so you will not have to reconfigure all the directives, only the ones that change.

As with the modules, there are two directories to contain the virtual sites, one for the available ones and one for the active ones: */etc/apache2/sites-available* and */etc/apache2/sites-enabled* respectively. The latter contains symbolic links to the sites of the former that are active.

This documentation is based on version 2.4 of apache, but as a help to those who have used version 2.2 or have websites configured with that version, I leave here some indications of the changes to be made. Actually none of them are directly from Virtual Hosts, but they are general directives that are usually used in this area.

Default site

The default virtual server configuration can be found at

```
gedit /etc/apache2/sites-available/000-default.conf
```

which will show us

```
<VirtualHost *:80>.
     # The ServerName directive sets the request scheme, hostname and port that
     # the server uses to identify itself. This is used when creating
     # redirection URLs. In the context of virtual hosts, the ServerName
     # specifies what hostname must appear in the request's Host: header to #
     match this virtual host. For the default virtual host (this file) this #
     value is not decisive as it is used as a last resort host regardless. #
     However, you must set it for any further virtual host explicitly.
     #ServerName www.example.com

     ServerAdmin webmaster@localhost
     DocumentRoot /var/www/html

     # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
     # error, crit, alert, emerg.
     # It is also possible to configure the loglevel for particular
     # modules, e.g.
     #LogLevel info ssl:warn

     ErrorLog ${APACHE_LOG_DIR}/error.log
     CustomLog ${APACHE_LOG_DIR}/access.log combined

     # For most configuration files from conf-available/, which are
     # enabled or disabled at a global level, it is possible to
     # include a line for only one particular virtual host. For example the
     # following line enables the CGI configuration for this host only
```

```
          # after it has been globally disabled with "a2disconf".
          #Include conf-available/serve-cgi-bin.conf
     </VirtualHost>
```

We can check that in the directory of active sites there is a file *000-default* which is the symbolic link to the default site.

Study and discuss in class the directives included in the default virtual site configuration.

In the file we can see that the root directory is */var/www/html* and the configuration that the directory has. Note that it is necessary to restart Apache for the configuration changes to take effect.

Note: until version 2.2 the `default` directory was */var/www* and this path is still very common. However, in

the main Apache configuration file, apache2.conf, we find:

```
     <Directory /var/www/html/>
          Options Indexes FollowSymLinks
          AllowOverride None
          Require all granted
     </Directory>
```

A directory inherits the directives of the parent directory if they are not overwritten. Therefore, all the directories that we create inside */var/www/* will have the same configuration as the root except `for` those directives that we specify in the directory's own configuration.

We are going to modify the default site configuration. Create a *DAW.html* file in the root directory with your name and your data. Now we want the index or home page of the directory to be the one you just created. What directive should you add? Is it possible to set more than one home page? If so, which one will load if we access the site without specifying a page? What happens if none of the pages are set as the home page? Look for a directive that disables the file listing even if it does not find any of the default pages.

Create a new directory inside the root directory. The name of the directory will be your last name. Create several html files but do not set any of them as the main one. Access the directory from a browser. Now modify the directory settings to list the contents. What happens? Set one of the files as the main file for the directory and try again.

Remember that after each change in the configuration it is necessary to restart Apache.

As you may have already seen in practice, to disable the listing of the contents of a server, virtual host or directory you use

```
Options -Indexes
```

If you want to activate it for a server, directory or virtual host you use

```
Options Indexes FollowSymLinks
```

The option to follow links is not mandatory but is usually added.

Note that it is necessary to check that the configuration is not overwritten in the configuration of the default virtual host or of some particular directory. What should be done in these cases is a check of the policy from the particular to the general: directories, virtual host and server.

Modifying error messages

The *ErrorDocument* directive is used to set custom error messages for different situations. The error codes that can be used are those defined by the w3 organization for the http protocol, in this case version 1.1, and can be consulted here. In this other page they are explained in a more understandable way. Note that the code is only the last number of each section and has three digits. This directive will be used as many times as necessary. It is set independently for each virtual site and is usually written under *CustomLog*.

In the previous section we have seen that an environment variable called *APACHE_LOG_DIR* is used to determine the directory where the log files will be located. In this installation it is */var/log/apache2/* but if not we can look for the files in the directory structure or consult the value of the environment variable.

Set custom actions for the 404 error. First redirect page not found queries to the site's home page. Then try redirecting these errors to the institute's home page. In the third test you should display a message directly that says "page not found, query First Name Last Name" with your first and last name. Finally create an html file explaining the error and have it displayed if it is the case.

Consult the list of defined errors and write down the ones you think are most useful. Discuss them in class.

Aliases to other directories

By default the structure of a web site will be the one we give to the directories from the mount point of the root directory of the site, however it is possible to include other directories from our server tree and make them look like part of the site as any other folder that is physically contained inside. It is something similar to having a symbolic link to files or folders in another directory.

In the main site configuration file seen above there is already an alias to a directory outside the folder defined for the site.

To perform this operation, the *Alias* directive is used and just below it (not mandatory but a good practice) it is configured with the directive *Directory* like any other directory.

In the quick guide to directives, all directives with "d" in the third column can be applied to directories.

Create a /var/extras directory and place an HTML page inside that allows you to differentiate it from the others. Configure the directory to appear as /extras on our web site and apply the directory directives you consider important.

Redirections

Sometimes it is useful to be able to redirect calls to a web address to be processed at another point. For example if we have changed the structure of our site and relocated the contact page we may want users accessing via the old address to be automatically redirected to the new one. It can also be useful if we have split our site for easier management or maintenance.

For this purpose, the *Redirect* directive is used, which allows us to associate an absolute or relative address to another.

Create a redirect on the main server site so that in case someone wants to access /hobbies it redirects to a page that you visit very often.

Now create a redirection that in case someone wants to access /ext will show the content of /extras that you created in the previous point.

Creation of virtual hosts

There are three different ways to create virtual hosts in Apache:

1. Name-based: This type is the most common option. Everything is configured so that multiple DNS domains point to a single machine with Apache. It requires DNS server configuration to work. This method makes it very easy to migrate our server to another IP address. It is the only method we are going to study.
2. IP-based: In this method you need to configure the IP addresses of each site in Apache. The physical server will have multiple IP addresses, one for each site.
3. Port-based. Each site will be served on the same IP or name but on different ports. It is an extension of any of the previous alternatives.
4. Several main servers: In this option several main configurations are maintained on the server. Its use is only recommended if it is necessary to have a different configuration file for each site. It is rarely used and is the least recommended option for configuring our server.

The use of these alternatives is not exclusive. Several or all of them can be combined in the same Apache server.

In this address you can find the documentation about apache virtual *hosts*. In this other one you can see configuration examples.

Study and discuss in class the first examples in the link above. Focus only on the first few, which deal with the types listed above.

The necessary directives to create virtual *hosts* are in the Apache *core* module so it is not necessary to activate any module.

To use these methods it is necessary to be able to make all the domain names associated to the virtual *hosts* that we are going to create point to the IP address or IP addresses of our web server. If we have contracted a domain name with a provider they will provide us with a way to do this.

In case we have or want to publish our own sites we must configure a DNS server. In the appendices you will find a very simple way to configure DNS on our server. Another option to test it is by editing the *hosts* file

```
sudo gedit /etc/hosts
```

and in it add the necessary aliases.

Virtual sites use a directory structure similar to the one we have seen for modules. At

```
/etc/apache2/sites-available/
```

We find the available sites, which, when activated, create an alias in

```
      /etc/apache2/sites-enabled/
```

Name-based virtual hosts

This method is the most recommended as it requires a single IP address to host multiple sites. We can see the specific Apache documentation.

NOTE: In Apache, until version 2.2 it was necessary to add the following.

We can check that the */etc/apache2/ports.conf* file contains the following directive

```
      NameVirtualHost *:80
```

This directive is required to be able to use name-based virtual hosts. When enabled it disables the main server (the one we had used in the manual installation) and therefore in this type of installation the virtual server is actually a virtual site by default.

If you receive an error "*Apache2 NameVirtualHost *:80 has no VirtualHosts*" it is usually because you have put the NameVirtualHost directive in both site configuration files. Since it can be a server directive, putting it in the *httpd.conf* is fine.

END NOTE.

The first step is to create the DNS records for the domain name to point to our machine or add them in the *hosts* file.

We must create a directory for each site in order to have separate documents. In addition, an *index.html* file should be created in the directory so that it loads when the site is consulted.

```
mkdir /var/www/example2.es
```

At this point we can access the data of example2 through the address http://localhost/ejemplo2.es/ but our goal is to be able to do so by typing http://ejemplo2.es/.

The third thing we should do is to write the data of the new virtual host in the Apache configuration. I remember that we have seen two different cases of where the Apache configuration was located, but with the option we are currently using the main configuration was in the */etc/apache2/apache2.conf* file and many others were imported from it. We can see that a complete folder is imported from which it takes the configuration of the virtual hosts.

```
...
# Include the virtual host configurations:
Include sites-enabled/
...
```

However, before doing this we must create the virtual host in the available hosts folder and we will activate it when it is finished. The folder where we can add the data of our virtual hosts is */etc/apache2/sites-available*

If we go to the folder indicated we can consult the configuration file of a virtual host, the default site. This can serve us as an example and guide to create others.

Create the configuration file

```
sudo gedit /etc/apache2/sites-available/example2.en
```

and write the following

```
      <VirtualHost 10.0.2.15:80>
           ServerName ejemplo2.es
           ServerAlias www.ejemplo2.es
           ServerAdmin alguien@ejemplo2.es
           DocumentRoot /var/www/example2.en
           #
           # Other directives can go here.
           # By default it inherits those of the main
           file. #
      </VirtualHost>
```

Among all the directives that can be included, those referring to directories are especially useful, since if the site consists of several (the most common) it may be of interest to have different configurations for each one.

It is also common to configure separate error logs for each site using the appropriate directive. This is done within the configuration of each virtual site leaving the main one for the server itself. We have already seen that in the default site they were included.

```
           ErrorLog ${APACHE_LOG_DIR}/error.log
           CustomLog ${APACHE_LOG_DIR}/access.log combined
```

But of course we must give other names to our records so that the same ones are not used.

It is a good idea to review the main site configuration to get an idea of what to configure as in the example above I have reduced the directives to a minimum.

If instead of the IP address we write * it will be done for all the IPs that the machine has.

In the two cases where we have written the IP address and the port we could have put an asterisk (*) instead of the address but we would be telling Apache to listen to all requests which could generate conflicts. It is always preferable to specify the data.

Now we have configured the new host so we can activate it.

```
    sudo a2ensite example2.es
```

and reload Apache sites

```
    service apache2 reload
```

if we get the error message

```
    * Reloading web server config apache2                                    apache2: Could not
reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName
```

We need to open the *httpd.conf* file and add

```
    ServerName LocalHost
```

Or the full name of our server.

We can now access our document through www.ejemplo2.es

To deactivate a virtual site is done with (e.g. to deactivate the default site).

```
sudo a2dissite default
```

Create a new virtual host based on name and test that everything is correct. Create another one and you will see that you now have the ability to access three sites on the same machine. Make the association via the Linux *hosts* file. Then remove the entry in the hosts file and make the association via a DNS server. Finally disable the virtual site.

For this practice you will need to configure your DNS server. In the appendices I explain how.

Access control

At this point we start with several aspects related to the security of our web server.

With this first section we are going to consider the aspects that will allow us to filter access to certain resources.

Access control refers to any method that allows us to filter access to a given resource. There are mainly two modules involved in access control in Apache: mod_auth_core and mod_authz_host although mod_setenvif and mod_rewrite are also used to cover all a s p e c t s .

There are also three methods for managing access control related to these modules. Access control is closely related to authorization and authentication, which we will see in the next section.

Address-based access control

This type of control is based on the use of the mod_authz_host module and the IP addresses of the machines that want to access our server. We can check that this module is already active in our installation and in fact we have already used this type of access control to allow access to parts of our server from the host machine in the practices.

We already commented at the beginning of the topic that for this case the directives that were used until version 2.2 were *Allow* and *Deny*. Generally they are associated with another one, *Order*.

However, nowadays everything is done with the *Require* directive. In the documentation itself we can find many examples and different uses but this change has greatly simplified the task. However, we must first focus on understanding how the directive works in a more generic way by studying the documentation of the moth_auth_core module.

Note that the same policy is used to allow or deny access by using *granted* or *denied.*

Investigate how to filter an entire domain and a range of IP addresses instead of single machines.

Is it possible to filter addresses with a mask that is not a multiple of 8? How?

What are containers for? Study RequireAll, RequireAny and RequireNone.

Create a directory with your name on the default site. Allow access from the host machine but deny access from the machine where the server is located.

Note that these directives are not exclusive to directories. They can be applied to the entire web site for example. For this they will go in the general configuration of the site and not within *Directory* tags.

Access control by environment variable

This type of access control is performed by using the mod_authz_host and mod_setenvif modules. It is based on allowing access according to the configuration of some environment variable of the user's machine and is therefore not highly recommended.

Access control with the rewrite module

By using the mod_rewrite module we can control access according to arbitrary criteria. For example, if we want to deny access during the period from 8:00 p.m. to 6:00 a.m., we would write

```
      RewriteEngine On
      RewriteCond %{TIME_HOUR} >20 [OR]
      RewriteCond %{TIME_HOUR} <07
      RewriteRule ^/fridge - [F]
```

The use of this method is based on the RewriteCond and RewriteRule directives, but is beyond the scope of this course and is provided only as an introduction to the possibilities it allows.

Authentication and authorization

These two terms are linked but are not the same thing, although many people confuse them. Authentication is checking that someone is who they say they are while authorization is checking that someone has permission to access a particular place or resource.

For example, if you want to travel abroad (outside the countries that have signed the Schengen Agreement) you need a passport and a visa. The passport is a general document that serves to prove that you are who you say you are, while the visa authorizes you to visit a specific country.

In computing, authentication can give us access to different resources for which we are authorized, and even these authorizations can vary depending on different circumstances. For example, we may have permission to access a certain resource in a certain time zone or from the office but not from home.

The authorization process usually involves authentication. For example, if a client wants to access a certain resource, the server asks the client to authenticate (by means of a 401: Authoritation Required status message), for example with a user and password request. If this authentication is positive, access will be allowed, otherwise it will be answered with another 401 message.

One problem with this form of authentication is that the password is neither encrypted nor hidden, so we will discuss the protocol later.
https. Anyone with a *sniffer* could intercept usernames and passwords.

In Apache we can focus on the documentation about these concepts. We will go through the whole link step by step and understand all the information provided. Especially interesting is the section on *Require*.

The *htpasswd* command is one of the first to appear. Find out how to use it and what it does. Since this version of Apache it is necessary to install it separately as it is not included in the general installation.

sudo apt-get install apache2-utils

The command can encrypt passwords in different ways.

In Apache since version 2.3 the authentication base has been centralized in mod_authn_core and the authorization base in mod_authz_core. The number of modules involved is actually much larger, but they are grouped into three large sets: One for authentication, one for authorization, and one for determining against which system the credentials will be checked in authentication (e.g. a file, a database, etc).

Modules mod_auth_core and mod_auth_basic

These modules allow us to perform authorization in a fairly basic way. They are going to allow us to establish access by means of user and password to sections of our site. Basically it is what we have commented in the previous link.

For example we can set a directory that must be accessed with a password. We can choose any directory as long as the user that launches apache (*User* directive) has access to it. It is not necessary that the directory is in the basic structure of our site, but if it is not we will have to establish an *Alias*.

```
mkdir /var/www/private
mkdir /var/secret
```

Now we add to the site where we want to manage authentication

```
gedit /etc/apache2/sites-available/000-default
```

the configuration for the directory

```
<Directory "/var/www/private">
        AuthName "Private access: Enter your username and password" AuthType
        Basic
        AuthUserFile /var/secret/.members
        Require valid-user
</Directory>
```

- *AuthName* tells the user what to do. It is a message to the user.
- *AuthType* is the authentication type we will use; http only supports *Basic*. The other option that exists is *Digest* which unlike the *Basic* option does not transmit usernames and passwords as plain text (and is therefore a better security option) but is not available for all Web browsers as an out-of-the-box option. The encryption used by the *Digest* option is quite weak and although its use is not identical to that of the *Basic* option it is quite similar, so we will not look at it.
- *AuthUserFile* is the file that will be used to store the passwords. It indicates the path and will be named .members.
- *Require* specifies that it will be necessary to log in with a valid user.

Now we need to create the password file.

```
htpasswd -c /var/secret/.members srsergio
```

Where you must specify the path to the apache executables if it is not the same, -c is the option to create the file so if you are going to add another member you must remove it. The path to the password file must be the same specified for the site configuration and *srsergio* will be the username we want to create. It will ask us to create a password.

We can make sure that the user has been created by opening the file. Note that the password is encrypted.

```
gedit /var/secret/.members
```

Restart apache

```
apachectl restart
```

Now you have a section for access only with username and password.

Note that the weakness of http in the transmission of information makes this method unsafe but it can be solved by using https as we will see later.

Create a private section on your site and enter some html files. Create at least two users who have access and try logging in with both them and invalid data.

We always talk about the fact that information transmitted over the Internet as plain text is very easy to intercept. How would this be done in our internal network? What programs do you need? Would you be able to intercept a username and password sent by a user to access the private section created in the previous point?

Extension: Investigate how to do the same with the *Digest* option. It seems that Apache has Digest disabled by default and all it does is to encode but not encrypt the data, so it is not even worth activating it.

The .htaccess files

Although we have used them intuitively, it is a good time to delve into the Apache configuration files in which we can use different sections.

The solutions seen so far are not very suitable if we want to be able to delegate the creation and control of private areas for specific members. This can be very useful if, for example, we have set up a web site for a company that has its own system administrator, since it will prevent us from having to manage everything ourselves. It will also make our work easier if there are many changes in the private zones or members connecting to them.

We can learn more about *htaccess* in this article. Another website introduces us to some of the most common uses of these files with a second part.

To allow the use of *.htaccess* files in our server or virtual site (the directive can be used in both environments) the first thing we must do is to modify the configuration file. In my case I am going to do it in the default virtual site.

We must modify the directive

```
    AllowOverride None
```

And change it to

```
AllowOverride AuthConfig
```

Where to modify the directive depends on what we need. Keep in mind that directives are inherited if another more specific one is not found, therefore, in our virtual site configuration file by default it should go in the /var/www directory at least since if we put it in the root but not in the first one, the previous configuration would be kept.

```
<Directory /var/www/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride AuthConfig
    Require all granted
</Directory>
```

This allows us to modify the authorization directives through an *.htaccess* file. In many sites they indicate that it is necessary to allow the overwriting of all the directives by means of *AllowOverride All* but it is evident that it is worse option.

Then we restart the Apache server

```
apachectl restart
```

Now we could create directory files and configure their access control in each one of them.

```
mkdir /var/www/files/ cd
/var/www/files/
```

In each directory that we want to manage this way we must create a *.htaccess* file and give it a content similar to the following.

```
AuthName "Private Section: .htaccess Test" AuthType
Basic
AuthUserFile /var/secret/.members
Require valid-user
```

Please note that:

- What we have done is to move the access control settings out of the server/virtual site configuration file into a separate file.
- I have used the same file of users and passwords as in the previous point to give coherence to the examples, but this is not necessary.
- Users and their passwords would be created as in the previous section.
- The creation or modification of such a file does not involve restarting the server.
- Security should be improved by changing the access permissions to the *.htaccess* file. At least the Apache user must have access.

Create two directories in your site by default and configure them so that two users can access the first one and only one of them can access the second one. To do this, *Require* must contain your user name.

Grouping users for access control

If we want to further refine the access control we have several alternatives. For example we can allow access to only some users by specifying their names in the *Require* clause wherever we use it.

```
      require user sergio maria
```

Another alternative is to create different user files for different directories of our site.

```
      <Directory "/var/www/sales">
              AuthName "Private access: Enter your username and password" AuthType
              Basic
              AuthUserFile /var/secret/.members-sales
              Require valid-user
      </Directory>

      <Directory "/var/www/finance">
              AuthName "Private access: Enter your username and password" AuthType
              Basic
              AuthUserFile /var/secret/.members-finances
              Require valid-user
      </Directory>
```

The last option is to use the *AuthGroupFile* directive. This way we can have all the users in the same file and then assign groups to them in another file.

```
      <Directory "/var/www/sales">
              AuthName "Private access: Enter your username and password" AuthType
              Basic
              AuthUserFile /var/secret/.members
              AuthGroupFile      /var/secret/.groups
              Require group sales
      </Directory>
```

```
<Directory "/var/www/finance">
        AuthName "Private access: Enter your username and password" AuthType
        Basic
        AuthUserFile /var/secreto/.members
        AuthGroupFile      /var/secret/.groups
        Require group finances
</Directory>
```

Please note that:

- The same member file is used for all directories.
- The policy and the group file must be added.
- In the Requires clause you specify the group you can access.

The .groups file will be a text file with something like

```
sales: sergio maria
finance: carlos roberto
```

Modify the configuration of the previous exercise so that two different user `groups` can access each directory with the last method. Can a user be in more than one group?

Add a third directory and another group. Give access to the third directory to two of the three groups. How is this done?

The HTTPS protocol

We have already seen that the HTTP protocol is very insecure for transmitting sensitive information. It is very easy to capture this information and read data such as usernames or passwords. Any information that is sent is unencrypted so it is transmitted as text that anyone who captures our communication can read.

The HTTPS protocol is used to avoid this problem. Its acronym stands for *Hypertext Transfer Protocol Secure*, which already indicates that it adds security to HTTP. It is not really a protocol, but is based on introducing a layer with the SSL protocol (or derivatives, more on this later) between the transport layer and the application layer in the TCP/IP protocol suite.

The HTTPS protocol provides user authentication with the server to which it connects. For example, when connecting to

our webmail (e.g. Gmail) is usually done through an HTTPS connection.

With the use of this protocol we also get encryption of the information we send and receive. Therefore, when we connect to a server that only uses HTTPS for authentication, the browser warns us (once we have connected) that the rest of the communication does not use HTTPS and therefore the transfer of information may be insecure. This protects us against *Man-In-The-Middle* attacks. Everything sent in an HTTPS message is encrypted, including the headers. Of course the encryption is also subject to attacks that attempt to crack the key and algorithm used to encrypt the content.

Research what algorithms and what kind of keys are currently used on the Internet.

What are symmetric key algorithms and asymmetric key algorithms? What about private keys and public keys? How are they used?

This Wikipedia page can help you understand these concepts and the ones we will see below. You will need to have an idea of the previous points of this exercise to understand it.

A site using HTTPS should have all its contents protected by this protocol to avoid possible attacks or information theft through the insecure parts.

HTTPS is slightly less efficient than HTTP, so if it is used on sites where the transfer of information is very large it can be noticeable in performance. Of course, if the information transferred is sensitive, the advantages outweigh the disadvantages.

The HTTPS protocol uses port 443 by default. As was the case with HTTP and port 80, if this port is used, it is not necessary for the client to indicate it in the browser address bar.

Digital Certificates

The HTTPS protocol encrypts the communication so that whoever captures frames from it cannot see the contents. Current web browsers base the use of HTTPS on the knowledge o f Certificate Authorities that issue Digital Certificates to ensure that the server we connect to is who it claims to be. These Certificate Authorities are either dedicated agents or companies such as Microsoft.

The use of HTTPS is based on the trust provided by the entities that issue the certificates. When we use a particular web browser, the company that has developed it has already introduced in it certain Certificate Authorities that it considers trustworthy. These entities can be consulted in the browser itself and can be modified. In the image you can see part of t h e list included in Firefox.

Investigate in the different browsers you have installed how to consult the list of trusted Certificate Authorities.

In the screenshot you can see other tabs for the certificate manager, what is the purpose of each of them?

There are many servers that use certificates not issued by these authorities. In these cases it is up to the user to accept them or not. The browser displays a warning when we are going to connect and we will have to decide whether we want to continue or prefer not to connect to the destination.

In the image we can see an example with several parts:

1. If we do not trust the destination we must press the button to exit.
2. We can get more information. In this case we see that the entity is gob.es so we can decide to trust it.
3. If we want to trust, we must read the risks involved.
4. We can add a security exception to trust this site from now on.

What is the digital signature and what is its relationship with digital certificates?

A Digital Certificate is an electronic document that links a public key with a Digital Signature and personal information about the person or organization that wants to use the public key. It serves to ensure that the key belongs to that person or organization. The personal information attached is usually the name, address, email address, etc. The Digital Signature is usually that of a recognized Certification Authority so that customers can trust that the Public Key and the personal information correspond to the same person/organization. This type of certificates are known as Root Certificates since they are at the highest level of a certificate tree. The user must trust the entity issuing the Root Certificate as it is the one that ensures the authenticity of all certificates issued by it. The list of certificates that are included by default in a web browser fall into this category and the user relies on the browser developers to ensure that these certificates are trusted.

As you can see Internet security is based on many levels of trust! No one should panic. The authorities including the major browsers are verified and regularly checked.

In this Wikipedia page you can get much more information about Digital Certificates. In this image you can

see the use of digital certificates.

Certificate Servers are responsible for validating or certifying the keys.

Several Digital Certificate formats are currently used on the Internet. The most widespread is X.509.

Read this page and comment in class on the points you find most interesting about the information obtained.

In our browser we can consult the details of a certificate. If we click on the certificate in question and click "View", a "Details" option usually appears where we can consult data such as the algorithm and the signature value.

Obtain a digital certificate

In order to use HTTPS on our server it is necessary to have a Digital Certificate. We can obtain a Digital Certificate from a *Certificate Authority* (*CA*) or create our own Certificate Authority and generate our own Digital Certificates.

In Spain, we can obtain Digital Certificates through the Fábrica Nacional de Moneda y Timbre or through companies such as Verisign.

To obtain a Certificate from a Certificate Authority we generally have to prove that we are who we say we are (as established by each authority) and generate a request for our server (*certificate signing request, CSR*) that must be sent to the authority. When our request has been accepted, we can install the certificate on our server.

SSL/TSL

Secure Socket Layer (SSL) was developed by Netscape in the 1990s. There have been three versions of SSL but currently 3.0 is used although we should stop using it as it is an obsolete protocol with known vulnerabilities. It has evolved into another *Transport Layer Security* (TLS) protocol that also has three versions, 1.0, 1.1 and 1.2. Most current browsers use version 1.0.

As mentioned above, HTTPS is an implementation of HTTP over SSL or TSL on the server. This protocol is placed on

top of the transport layer where the two most typical protocols are TCP and UDP.

It is a protocol used to ensure confidentiality, authenticity, integrity and non-repudiation between the client and the server.

Research and comment on the four concepts in the previous paragraph.

There are two modes: One in which only the server proves its identity and another in which both the client and the server use Digital Certificates.

The applications of SSL/TSL use are multiple and include the creation of virtual private networks (VPN), use in e-commerce and e-mail, etc.

HTTPS in Apache

Apache uses a specific module based on a project called OpenSSL; despite the name it also implements TLS. To use HTTPS in Apache, the *mod_ssl* module must *be* active.

```
sudo a2enmod ssl
service apache2 restart
```

Now the server should be listening on both port 80 (http) and 443 (https). If we look at the port configuration file

```
gedit /etc/apache2/ports.conf
```

in which we see the different files and that we could use GNUTLS.

```
Listen 80

<IfModule ssl_module>
     Listen 443
</IfModule>

<IfModule mod_gnutls.c>
     Listen 443
</IfModule>
```

we will see that if the module *mod_ssl* is active, the command

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change #
    the VirtualHost statement in /etc/apache2/sites-available/default-ssl #
    to <VirtualHost *:443>.
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
    Listen 443
</IfModule>
```

We have seen that this Apache `distribution` comes with two `default` sites. The one we have not used is called *Default-SSL*. If we activate it we would already have a site with this configuration that would listen via secure connection.

```
sudo a2ensite default-ssl
service apache2 reload
```

Now we would have available the possibility to connect securely to both servers. If we do not specify the protocol or we use http it will connect in the standard way. However, if we connect using https it will show us a security exception like the one we saw before. This is because when installing Apache a self-signed certificate is created for the site by default.

If we open the site configuration file with ssl we can see how it is configured.

```
        #A self-signed (snakeoil) certificate can be created by installing
```

```
            #the ssl-cert package. See
        #/usr/share/doc/apache2    .2-common/README.Debian.gz for more info.
            #If both key and certificate are stored in the same file, only the
        #SSLCertificateFile directive is needed.
        SSLCertificateFile     /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

After performing the above steps, `try` intercepting frames from http and https connections. What differences do you see? Save a frame of each type to a file for discussion in class.

If you notice, the two sites by default have the same directory configured as the root directory for the documents. What would happen if you change the root directory of one of the two. Try it.

It is a bad option to leave open the possibility of accessing the same site through a secure and an insecure connection. Therefore, if we enable a site with HTTPS we should not have the equivalent with HTTP active as it happens with the two by default. The correct thing to do would be to disable the default.

If we disable the default site we can see that now, if we do not specify the protocol, it uses HTTPS by default.

```
    sudo a2dissite default
    service apache2 reload
```

Study the rest of the default site configuration file with ssl and explain what the other directives do. The links below contain all the necessary information.

https://httpd.apache.org/docs/2.4/en/ssl/

https://httpd.apache.org/docs/2.4/mod/mod_ssl.html

Creating a virtual site with HTTPS

Now that we have tested the `default` secure site, let's create one from scratch. I will leave the default site enabled to make an example with one site with HTTP and another with HTTPS.

```
sudo a2dissite default-ssl
sudo a2ensite default
service apache2 reload
```

As we have seen before, in order to use SSL in Apache it is necessary to have a certificate. The one installed for the example site is no longer valid and we have to get one. We can purchase one from a CA or create a self-signed one. For obvious reasons we will use the latter option.

To obtain a certificate it is necessary to generate a private key and for that we need a domain name, so the first thing to do will be configure the DNS correctly. In my case I am going to call it www.con-ssl.es and it will have to be configured as a virtual host by name.

To generate the key, use the command genrsa

```
openssl genrsa -out clavepru.key 2048
```

You can generate a password for the key, but if you are going to use it to create a certificate it is not a good idea because every time the web server needs to access the key you will have to enter the password. If we do not mind entering the key every time the server is restarted, we would use the option

```
openssl genrsa -des3 -out clavepru.key 2048
```

Minimum key lengths of 2048 bits are currently recommended.

The following will be to generate a request for our certificate. This request is the one we should send to the CA to obtain a certificate signed by them. Then we would wait for them to sign it and send us the certificate to install it. We will use a self-signed one.

```
openssl req -new -key clavepru.key -out peticionpru.csr
```

When we execute this command we are asked for information of the company for which it is the certificate. We fill it in until we finish. The fields ending in [ ] are not mandatory. If we had generated the key with password it would ask for it before filling in the information.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
```

```
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
------
Country Name (2 letter code) [AU]:EN
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SergioCuesta
Organizational Unit Name (eg, section) []:SC
Common Name (e.g. server FQDN or YOUR name) []:Sergio Cuesta
Email Address []:una@con-ssl.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A password challenge []:
An optional company name []:
```

To obtain a self-signed certificate we will use the command

```
openssl x509 -req -days 365 -in requestpru.csr -signkey keypru.key -out certificatepru.crt
```

Which indicates that it will use the X.509 format and will be valid for one year. If all goes well we should see something like the following

```
Signature ok
subject=/C=EN/ST=Madrid/L=Madrid/O=SergioCuesta/OU=SC/CN=Sergio Cuesta/emailAddress=una@con-ssl.es
Getting Private key
```

In many places you will see that instead of *crt* files are created with the *pem* extension. In theory the difference is that *crt* only contains the certificate while *pem* contains both the certificate and the key but in practice this is ignored and it is the same to use one or the other.

Note that all the files that have been created in the previous steps have been generated in the directory in which we were, so you have to move them to the appropriate places. The request is not necessary.

```
sudo mv clavepru.key /etc/ssl/private/ sudo
mv certificadopru.crt /etc/ssl/certs/
```

We will create a directory for the content of the secure site

```
mkdir /var/www/con-ssl/
```

We proceed to create the virtual site by name

```
<IfModule mod_ssl.c>

NameVirtualHost 192.168.1.36:443

<VirtualHost 192.168.1.36:443>
     ServerName con-ssl.es
     ServerAlias www.con-ssl.es
     ServerAdmin alguien@con-ssl.es
     DocumentRoot /var/www/con-ssl

     <Directory /var/www/con-ssl>
           DirectoryIndex index.html
```

```
                Options -Indexes
                AllowOverride None
                Require all granted
           </Directory>

           ErrorLog ${APACHE_LOG_DIR}/error_con_ssl.log
           LogLevel warn
           CustomLog ${APACHE_LOG_DIR}/con_ssl_access.log combined

           # This is the part of SSL
           SSLEngine on

           SSLCertificateFile /etc/ssl/certs/certificadopru.crt

           SSLCertificateKeyFile /etc/ssl/private/clavepru.key

           # Remember that the following is to maintain compatibility with certain #
           versions of Microsoft Internet Explorer
           BrowserMatch "MSIE [2-6]" \
                nokeepalive ssl-unclean-shutdown
                downgrade-1.0 force-response-1.0

           BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
      </VirtualHost>

      </IfModule>
```

We enable the site and reload Apache

```
      sudo a2ensite con-ssl.es
```

```
    service apache2 reload
```

Remember that the name we have associated in the DNS goes to the IP address of the machine so we will have to use https://www.con.ssl.es to access the secure site. If we use HTTP it goes to the default site. To avoid this we should add another entry to the DNS to go to the site without SSL (port 80) and modify the virtual host configuration to respond to requests to that other domain name instead of all (*).

If we have acquired the certificates through a trusted CA, the warning will no longer appear when a client connects. Evidently for a professional site it is more than recommendable.

For cases where we are involved in the deployment in a very large company or for example a university, we may be interested in creating our own CA for our own use. It is also possible to create a site where the clients have to access using their own certificate, for example in the Tax Agency. Both of these things can be done using *OpenSSL* but it is totally beyond the scope of this course.

Since the discovery of the Poodle attack, it has become even more important to correctly configure the version of encryption we use.

Create two different sites, one with HTTPS and one without. Configure everything correctly so that when going to one site or the other it uses the appropriate protocol without the need for the user to specify it in the address bar of the web browser.