

HTTP - HTTPS

Despliegue de Aplicaciones Web

Francisco Javier Arruabarrena Sabroso
Erin Francisco Sapeg Soriano

Índice

Supuestos previos	2
Parte 1: HTTP	3
Tareas de administración básica del servidor web	3
Comprobar el estado del servidor	3
Identificar la versión de apache instalada	3
Identificar el fichero de configuración principal y ajustar parámetros.....	4
Gestión del log de accesos y del log de errores	4
Configurar mecanismos de autenticación y control de acceso del servidor web	5
Creación de directorios y archivos adicionales	5
¿Qué es un módulo de Apache?	6
Habilitar autenticación básica.....	8
Bloqueo por IP	11
Análisis de peticiones y respuestas GET y POST	12
Implementación del código PHP	13
Análisis detallado del código PHP.....	13
Realizar solicitudes GET y POST	15
Conclusiones.....	17
Parte 2: Implementación de HTTPS con certificados autofirmados	18
Abrir el puerto 443	18
¿Por qué utilizar un certificado autenticado?	18
Generar el certificado de autofirmado	18
Configurar Apache para usar HTTPS	19
Analizar el tráfico con WireShark	20
Configurar la redirección de HTTP a HTTPS	23

Supuestos previos

Se van a tener en cuenta los siguientes supuestos previos:

1. Arquitectura del sistema:

- Un ordenador anfitrión que actuará como cliente.
- Una máquina virtual Ubuntu desktop que funcionará como servidor web y servidor de aplicaciones.
- Otra máquina virtual Ubuntu desktop que servirá como servidor de base de datos.

2. Configuración del servidor web y de aplicaciones:

- Sistema operativo: Ubuntu Desktop
- Servidor web: Apache instalado
- Lenguaje de programación: PHP instalado
- Cliente MySQL instalado

3. Configuración del servidor de base de datos:

- Sistema operativo: Ubuntu Desktop
- Servidor MySQL instalado

4. Conectividad:

- Todas las máquinas pueden comunicarse entre sí en la red.

Parte 1: HTTP

Tareas de administración básica del servidor web

Comprobar el estado del servidor

Para comprobar el estado del servidor Apache, se utilizará el siguiente comando.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo systemctl status apache2
[sudo] contraseña para jarasa03:
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: >
   Active: active (running) since Fri 2025-01-24 16:13:30 UTC; 53s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 1203 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SU>
  Main PID: 1265 (apache2)
     Tasks: 6 (limit: 7082)
    Memory: 21.2M (peak: 21.6M)
       CPU: 151ms
    CGroup: /system.slice/apache2.service
           └─1265 /usr/sbin/apache2 -k start
             1270 /usr/sbin/apache2 -k start
             1271 /usr/sbin/apache2 -k start
             1272 /usr/sbin/apache2 -k start
             1273 /usr/sbin/apache2 -k start
             1274 /usr/sbin/apache2 -k start

ene 24 16:13:29 Jarasa03-Ubuntu-24 systemd[1]: Starting apache2.service - The A>
ene 24 16:13:30 Jarasa03-Ubuntu-24 apachectl[1242]: AH00558: apache2: Could not>
ene 24 16:13:30 Jarasa03-Ubuntu-24 systemd[1]: Started apache2.service - The Ap>
lines 1-20/20 (END)
```

Este comando muestra si Apache está activo (running) o inactivo (stopped). También dará información sobre cuándo se inició el servicio y su PID (Process ID).

Es importante comprobar el estado del servidor regularmente para asegurar de que esté funcionando correctamente y para detectar cualquier problema potencial.

Identificar la versión de apache instalada

Para identificar la versión de apache instalada se usará el siguiente comando.

```
jarasa03@Jarasa03-Ubuntu-24:~$ apache2 -v
Server version: Apache/2.4.58 (Ubuntu)
Server built: 2024-07-17T18:55:23
```

Este comando nos mostrará la versión exacta de Apache que estamos utilizando. Conocer la versión es crucial para la seguridad y el mantenimiento, ya que nos permite saber si estamos utilizando la versión más reciente y si necesitamos actualizarla.

Identificar el fichero de configuración principal y ajustar parámetros

El fichero de configuración principal de Apache en Ubuntu se encuentra en `/etc/apache2/apache2.conf`.

Para editarlo, se usará el editor nano con privilegios de superusuario.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo nano /etc/apache2/apache2.conf
```

Dentro de este archivo, se buscarán y ajustarán los siguientes parámetros:

1. **Timeout:** Este parámetro establece el tiempo máximo que Apache esperará para recibir o enviar una solicitud. Se ajustará a 300 segundos.

```
#  
# Timeout: The number of seconds before receives and sends time out.  
#  
Timeout 300
```

2. **KeepAlive:** Este parámetro permite que múltiples solicitudes se envíen a través de la misma conexión TCP. Se habilitará.

```
#  
# KeepAlive: Whether or not to allow persistent connections (more than  
# one request per connection). Set to "Off" to deactivate.  
#  
KeepAlive On
```

Después de hacer estos cambios, se guardarán los cambios (Ctrl + O, luego Enter) y saldremos del editor (Ctrl + X).

Gestión del log de accesos y del log de errores

Los archivos de log en Apache se encuentran generalmente en:

- Log de accesos: `/var/log/apache2/access.log`
- Log de errores: `/var/log/apache2/error.log`

Para ver el contenido de estos logs en tiempo real, se puede usar el comando 'tail' con la opción '-f'.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo tail -f /var/log/apache2/access.log
192.168.56.1 - - [31/Jan/2025:15:00:26 +0000] "GET / HTTP/1.1" 200 3460 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 OPR/115.0.0.0"
192.168.56.1 - - [31/Jan/2025:15:00:26 +0000] "GET /icons/ubuntu-logo.png HTTP/1.1" 200 3607 "http://192.168.56.101/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 OPR/115.0.0.0"
192.168.56.1 - - [31/Jan/2025:15:00:27 +0000] "GET /favicon.ico HTTP/1.1" 404 492 "http://192.168.56.101/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 OPR/115.0.0.0"
^C
jarasa03@Jarasa03-Ubuntu-24:~$ sudo tail -f /var/log/apache2/error.log
[Fri Jan 31 14:28:19.179103 2025] [mpm_prefork:notice] [pid 1211] AH00163: Apache/2.4.58 (Ubuntu) configured -- resuming normal operations
[Fri Jan 31 14:28:19.179690 2025] [core:notice] [pid 1211] AH00094: Command line: '/usr/sbin/apache2'
[Fri Jan 31 14:30:55.837127 2025] [mpm_prefork:notice] [pid 1116] AH00163: Apache/2.4.58 (Ubuntu) configured -- resuming normal operations
[Fri Jan 31 14:30:55.838585 2025] [core:notice] [pid 1116] AH00094: Command line: '/usr/sbin/apache2'
[Fri Jan 31 14:54:39.863022 2025] [mpm_prefork:notice] [pid 1116] AH00170: caught SIGWINCH, shutting down gracefully
[Fri Jan 31 14:54:39.931940 2025] [mpm_prefork:notice] [pid 3273] AH00163: Apache/2.4.58 (Ubuntu) configured -- resuming normal operations
[Fri Jan 31 14:54:39.931973 2025] [core:notice] [pid 3273] AH00094: Command line: '/usr/sbin/apache2'
```

Estos logs son cruciales para monitorear el tráfico del servidor y diagnosticar problemas. El log de accesos registra todas las solicitudes hechas al servidor, mientras que el log de errores registra cualquier problema que ocurra durante el procesamiento de las solicitudes.

Configurar mecanismos de autenticación y control de acceso del servidor web

Creación de directorios y archivos adicionales

Se ejecutarán los siguientes comandos uno a uno y en el respectivo orden.

Creación de un directorio, creando los padres si no existiesen:

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo mkdir -p /var/www/html/proteccionusuario
```

Similar al anterior, crea el directorio 'proteccionip' en la misma ruta:

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo mkdir -p /var/www/html/proteccionip
```

Cambia el propietario de directorio, aplicando el cambio recursivamente y estableciendo el usuario y grupo propietario a www-data (usuario estándar de Apache):

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo chown -R www-data:www-data /var/www/html/proteccionusuario
```


Similar al anterior, pero con el directorio 'proteccionip':

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo chown -R www-data:www-data /var/www/html/proteccionip
```

Cambia los permisos al directorio, lo hace recursivamente, estableciéndole permisos de lectura, escritura y ejecución para el propietario, y lectura y ejecución para grupo y otros.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo chown -R 755 /var/www/html/
```

Imprime el texto entre comillas y lo envía al siguiente comando (tee), escribe la entrada estándar en el archivo especificado con privilegios de superusuario. Esto crea un archivo PHP simple en el directorio 'proteccionusuario'.

```
jarasa03@Jarasa03-Ubuntu-24:~$ echo "<?php echo 'Este contenido está protegido por usuario y contraseña.'; ?>" | sudo tee /var/www/html/proteccionusuario/index.php
```

Similar al anterior, pero crea un archivo PHP en el directorio "proteccionip".

```
jarasa03@Jarasa03-Ubuntu-24:~$ echo "<?php echo 'Este contenido está protegido por restricciones IP.'; ?>" | sudo tee /var/www/html/proteccionip/index.php
```

Estos comandos preparan la estructura de directorios y archivos necesarios para las siguientes partes de la práctica, estableciendo los permisos correctos y creando archivos PHP de ejemplo.

¿Qué es un módulo de Apache?

Un módulo en Apache es un componente adicional que amplía las funcionalidades del servidor web. Apache, por diseño, es modular, lo que significa que su funcionalidad básica puede ser ampliada o personalizada mediante módulos. Estos pueden ser:

- **Módulos integrados:** Incluidos por defecto en Apache (por ejemplo, 'mod_dir', 'mod_mime').
- **Módulos opcionales:** Pueden ser habilitados o deshabilitados según las necesidades del servidor (por ejemplo, 'mod_ssl' para HTTPS o 'mod_rewrite' para reescritura de URLs).

Los módulos pueden añadir características como:

1. Autenticación y autorización.
2. Soporte para lenguajes de programación como PHP.
3. Compresión de contenido (gzip).

4. Reescritura de URLs.
5. Configuración de seguridad avanzada.

¿Por qué son importante los módulos?

1. **Flexibilidad:** Puedes habilitar solo los módulos que necesitas, manteniendo el servidor ligero y eficiente.
2. **Escalabilidad:** Permiten añadir nuevas funcionalidades sin modificar el núcleo del servidor.
3. **Seguridad:** Algunos módulos ofrecen características específicas para proteger el servidor (por ejemplo, 'mod_security').
4. **Compatibilidad:** Facilitan la integración con otras tecnologías (como bases de datos, lenguajes de backend, etc.).

¿Cómo saber qué módulos están habilitados y activos en tu sistema?

En Apache, se pueden listar los módulos habilitados utilizando:

```
jarasa03@Jarasa03-Ubuntu-24:~$ apache2ctl -M
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Loaded Modules:
  core_module (static)
  so_module (static)
  watchdog_module (static)
  http_module (static)
  log_config_module (static)
  logio_module (static)
  version_module (static)
  unixd_module (static)
  access_compat_module (shared)
  alias_module (shared)
  auth_basic_module (shared)
  authn_core_module (shared)
  authn_file_module (shared)
  authz_core_module (shared)
  authz_host_module (shared)
  authz_user_module (shared)
  autoindex_module (shared)
  deflate_module (shared)
  dir_module (shared)
  env_module (shared)
  filter_module (shared)
  mime_module (shared)
  mpm_prefork_module (shared)
  negotiation_module (shared)
  php_module (shared)
  reqtimeout_module (shared)
  setenvif_module (shared)
  status_module (shared)
```

¿Qué significa cada parte?

La primera parte antes de los paréntesis son los módulos cargados.

La parte entre paréntesis significa:

- (static). El módulo está integrado directamente en el binario principal de Apache y no puede ser deshabilitado sin recompilar Apache.
- (shared). El módulo está cargado dinámicamente desde un archivo .so y puede ser habilitado o deshabilitado fácilmente.

Habilitar autenticación básica

Habilitar el módulo auth_basic

Primero, se habilitará el módulo auth_basic. Este módulo proporciona autenticación básica HTTP. Es necesario habilitarlo para poder usar autenticación basada en usuario y contraseña en nuestro servidor Apache. Sin este módulo, no se podría implementar la protección por contraseña que se quiere.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo a2enmod auth_basic
[sudo] contraseña para jarasa03:
Considering dependency authn_core for auth_basic:
Module authn_core already enabled
Module auth_basic already enabled
```

Crear un archivo de contraseñas .htpasswd

El archivo '.htpasswd' almacena los nombres de usuario y contraseñas para la autenticación.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo htpasswd -c /etc/apache2/.htpasswd usuario1
New password:
Re-type new password:
Adding password for user usuario1
```

La opción '-c' crea un nuevo archivo. 'usuario1' es el nombre del usuario que se está creando.

Editar el archivo de configuración de Apache

Este archivo contiene la configuración del sitio web por defecto de Apache.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo nano /etc/apache2/sites-available/000-default.conf
```

Se edita para añadir las directivas de autenticación para el directorio específico que queremos proteger.

Se añaden las siguientes líneas:

```
# Líneas añadidas el 31/01/25 para añadir directivas de autenticación
<Directory "/var/www/html/proteccionusuario">
    AuthType Basic
    AuthName "Área Restringida"
    AuthUserFile /etc/apache2/.htpasswd
    Require valid-user
</Directory>
</VirtualHost>
```

El significado de cada línea es el siguiente:

- `<Directory "/var/www/html/proteccionusuario">`: Especifica el directorio al que se aplicarán estas reglas.
- `AuthType Basic`: Indica que se usará una autenticación básica HTTP.
- `AuthName "Área Restringida"`: Es el mensaje que verá el usuario en el cuadro de diálogo de inicio de sesión.
- `AuthUserFile /etc/apache2/.htpasswd`: Especifica la ubicación del archivo de contraseñas que se creen.
- `Require valid-user`: Permite el acceso a cualquier usuario válido en el archivo `.htpasswd`.

Después de todo esto se guardará el archivo y se saldrá del editor.

Reiniciar Apache

Es necesario reiniciar Apache para que los cambios en la configuración surtan efecto. Sin reiniciar, Apache seguiría usando la configuración anterior.

Resultado final

Después de completar estos datos:

- El directorio `"/var/www/html/proteccionusuario"` estará protegido por autenticación básica.
- Cuando alguien intente acceder a este directorio a través del navegador, se le pedirá un nombre de usuario y contraseña.
- Solo los usuarios definidos en el archivo `.htpasswd` podrán acceder al contenido de este directorio.

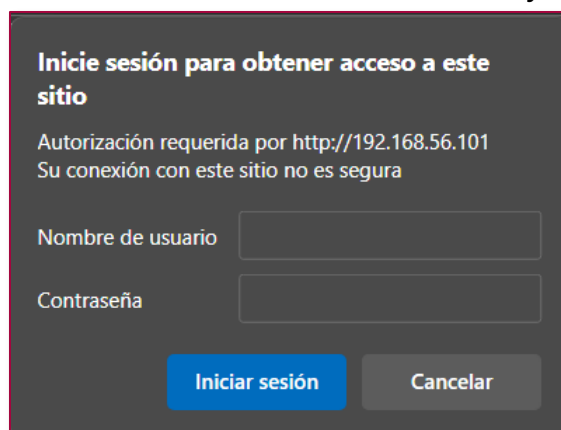
- Esta configuración proporciona una capa básica de seguridad para restringir el acceso a contenido sensible o privado.

Prueba del funcionamiento

Para acceder al directorio protegido, se pondrá desde el anfitrión en la url del navegador la ip del servidor seguido de /proteccionusuario/.

192.168.56.101/proteccionusuario/

Al poner ir a la url indicada, se solicitará usuario y contraseña.

A dark-themed login dialog box with a title "Inicie sesión para obtener acceso a este sitio". Below the title, it says "Autorización requerida por http://192.168.56.101" and "Su conexión con este sitio no es segura". There are two input fields: "Nombre de usuario" and "Contraseña". At the bottom, there are two buttons: "Iniciar sesión" (highlighted in blue) and "Cancelar".

Inicie sesión para obtener acceso a este sitio

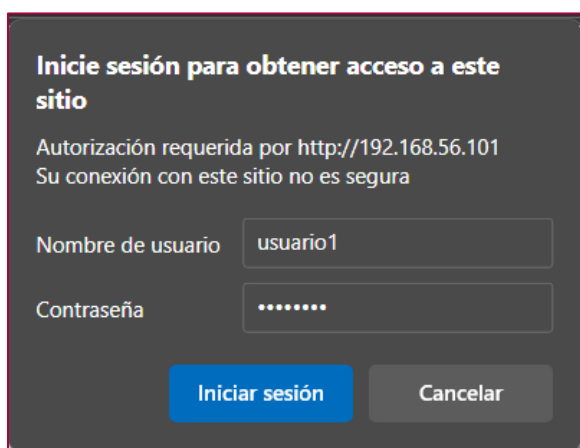
Autorización requerida por http://192.168.56.101
Su conexión con este sitio no es segura

Nombre de usuario

Contraseña

Iniciar sesión Cancelar

Se deberá poner un usuario y contraseña especificados anteriormente en esta documentación.

The same login dialog box as before, but now the "Nombre de usuario" field contains the text "usuario1" and the "Contraseña" field contains a series of dots, indicating a password is entered. The "Iniciar sesión" button remains highlighted in blue.

Inicie sesión para obtener acceso a este sitio

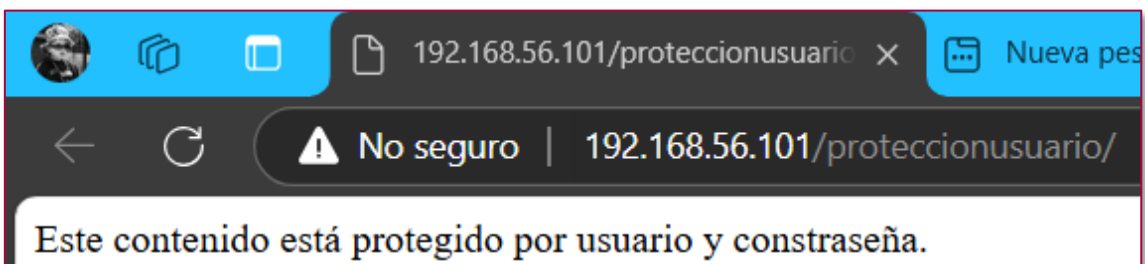
Autorización requerida por http://192.168.56.101
Su conexión con este sitio no es segura

Nombre de usuario

Contraseña

Iniciar sesión Cancelar

Al acceder se mostrará el mensaje que se puso con anterioridad de acceso a la web.



Bloqueo por IP

Para implementar el bloqueo por IP en Apache, se seguirán los siguientes pasos:

1. Editar el archivo de configuración de Apache.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo nano /etc/apache2/sites-available/000-default.conf
```

2. Dentro de la sección <VirtualHost>, se añadirá el siguiente código.

```
<Directory "/var/www/html/proteccionip">  
    Require all granted  
    Deny from 192.168.56.1  
</Directory>
```

3. Guardar el archivo y salir del editor.
4. Reiniciar Apache para aplicar los cambios.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo systemctl restart apache2
```

Explicación detallada

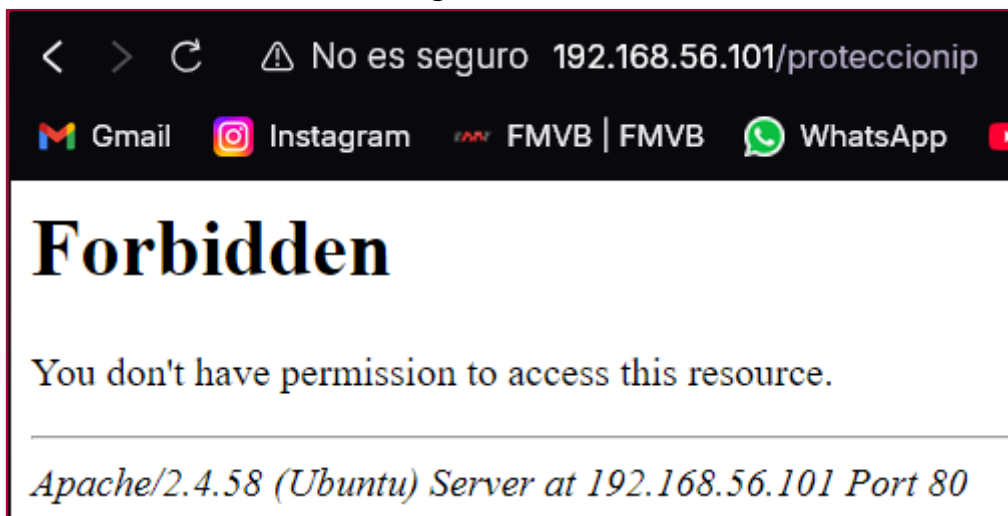
- La directiva <Directory> se usa para aplicar reglas a un directorio específico, en este caso "/var/www/html/proteccionip".
- Require all granted: permite inicialmente el acceso a todos los visitantes.
- Deny from 192.168.56.1 bloquea el acceso desde la IP especificada.

Prueba del funcionamiento

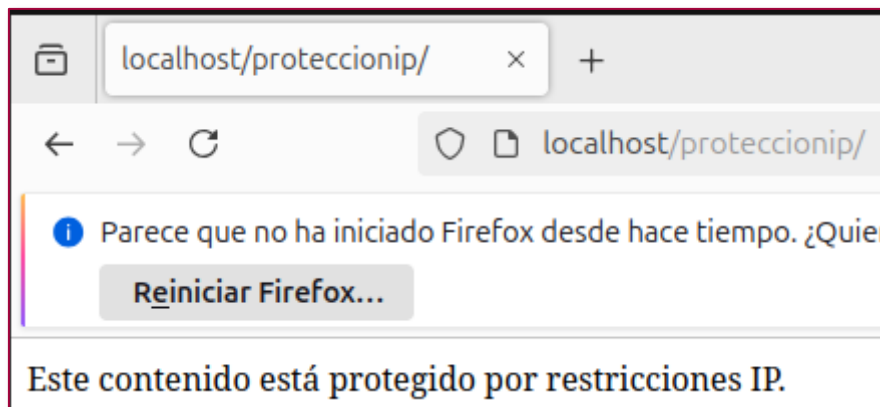
Para acceder al directorio protegido por ip, se pondrá en la url del navegador la ip del servidor seguido de /proteccionip.

```
192.168.56.101/proteccionip
```

Al acceder desde la ip denegada en el archivo de configuración de Apache, el acceso será denegado.



Si se accede desde cualquier otra ip el acceso será permitido.



Análisis de peticiones y respuestas GET y POST

Se va a implementar el siguiente código PHP, se averiguará para qué sirve y se realizarán peticiones GET y POST.

```
<?php
header('Content-Type: text/plain');

echo "Método de la solicitud: " . $_SERVER['REQUEST_METHOD'] . "\n";
echo "URI solicitada: " . $_SERVER['REQUEST_URI'] . "\n";

echo "\n--- Cabeceras de la solicitud ---\n";
foreach (getallheaders() as $key => $value) {
    echo "$key: $value\n";
}

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    echo "\n--- Parámetros GET ---\n";
    print_r($_GET);
} elseif ($_SERVER['REQUEST_METHOD'] === 'POST') {
    echo "\n--- Parámetros POST ---\n";
    print_r($_POST);

    echo "\n--- Cuerpo de la solicitud (RAW data) ---\n";
    echo file_get_contents('php://input');
}
?>
```

Implementación del código PHP

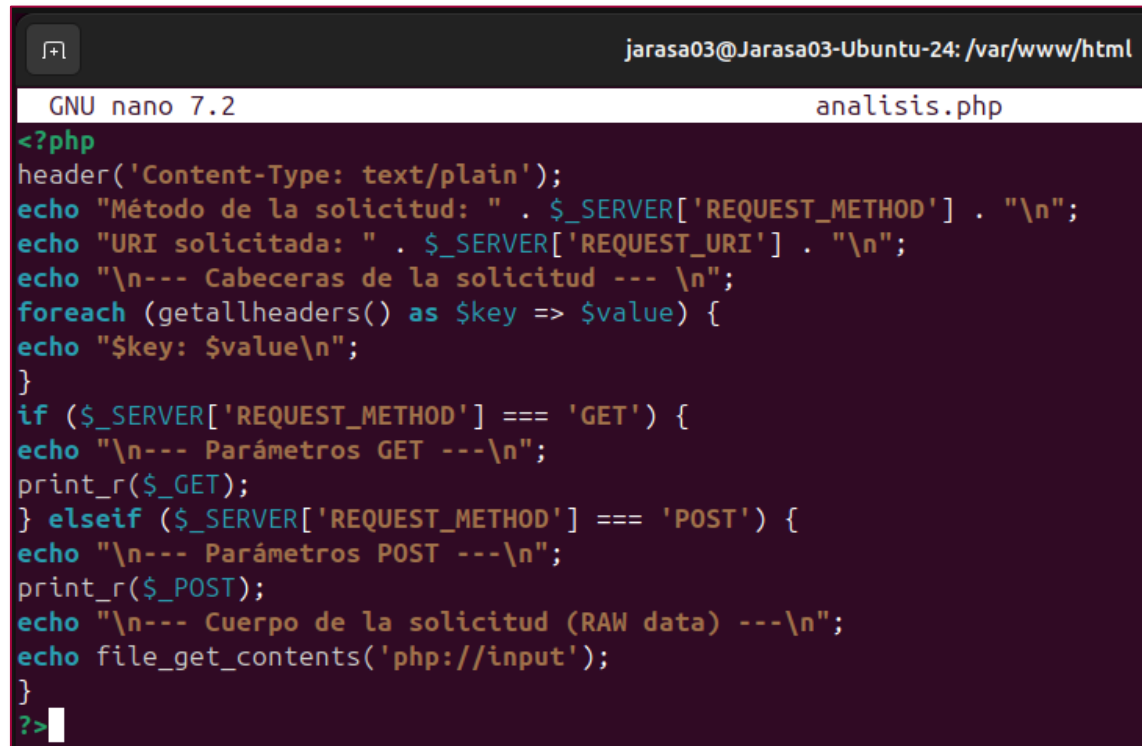
Navegar al directorio raíz de Apache

```
jarasa03@Jarasa03-Ubuntu-24:~$ cd /var/www/html/
```

Creación de un archivo PHP llamado “análisis.php”

```
jarasa03@Jarasa03-Ubuntu-24:/var/www/html$ sudo nano analisis.php
```

Introducir el código proporcionado



```

jarasa03@Jarasa03-Ubuntu-24: /var/www/html
GNU nano 7.2 analisis.php
<?php
header('Content-Type: text/plain');
echo "Método de la solicitud: " . $_SERVER['REQUEST_METHOD'] . "\n";
echo "URI solicitada: " . $_SERVER['REQUEST_URI'] . "\n";
echo "\n--- Cabeceras de la solicitud --- \n";
foreach (getallheaders() as $key => $value) {
echo "$key: $value\n";
}
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
echo "\n--- Parámetros GET ---\n";
print_r($_GET);
} elseif ($_SERVER['REQUEST_METHOD'] === 'POST') {
echo "\n--- Parámetros POST ---\n";
print_r($_POST);
echo "\n--- Cuerpo de la solicitud (RAW data) ---\n";
echo file_get_contents('php://input');
}
?>

```

Posteriormente, guardar el archivo y salir del editor.

Ahora, para asegurarnos de que el archivo tiene los permisos correctos:

```

jarasa03@Jarasa03-Ubuntu-24:/var/www/html$ sudo chown www-data:www-data analisis.php
jarasa03@Jarasa03-Ubuntu-24:/var/www/html$ sudo chmod 644 analisis.php

```

Análisis detallado del código PHP

1. `header('Content-Type: text/plain')`: Establece el tipo de contenido de la respuesta como texto plano. Esto asegura que el navegador interprete la respuesta como texto simple, no como HTML.

2. **echo "Método de la solicitud: " .**
\$_SERVER['REQUEST_METHOD'] . "\n": Muestra el método HTTP utilizado en la solicitud (GET, POST, etc). Esto es una variable superglobal de PHP que contiene el método de la solicitud.
3. **echo "URI solicitada :." . \$_SERVER['REQUEST_URI'] . "\n":**
Muestra la URI (Uniform Resource Identifier) solicitada. La variable mostrada contiene la ruta y los parámetros de la solicitud.
4. **foreach (getallheaders() as \$key => \$value) {**
echo "\$key: \$value\n";
}: getallheaders() es una función de PHP que recupera todas las cabeceras de la solicitud HTTP. Este bucle imprime cada cabecera y su valor.
5. **if (\$_SERVER['REQUEST_METHOD'] === 'GET') {**
echo "\n--- Parámetros GET ---\n";
print_r(\$_GET);
}: Si la solicitud es GET, muestra los parámetros recibidos en la URL. \$_GET es una variable superglobal que contiene todos los parámetros GET.
6. **elseif (\$_SERVER['REQUEST_METHOD'] === 'POST') {**
echo "\n--- Parámetros POST ---\n";
print_r(\$_POST);
echo "\n--- Cuerpo de la solicitud (RAW data) ---\n";
echo file_get_contents('php://input');
}: Si la solicitud es POST, muestra los parámetros POST y el cuerpo raw de la solicitud. \$_POST contiene los parámetros POST enviados. La última línea lee el cuerpo raw de la solicitud POST

Realizar solicitudes GET y POST

Solicitud GET

Se va a realizar una solicitud GET utilizando curl. Lo lógico sería hacerlo desde el cliente hacia el servidor, pero dado que la salida sería similar se ha hecho desde el servidor porque en Linux la salida del curl es más extensa y se puede explicar mejor.

```
jarasa03@Jarasa03-Ubuntu-24:~$ curl -v "http://192.168.56.101/analisis.php?param1=valor1&param2=valor2"
* Trying 192.168.56.101:80...
* Connected to 192.168.56.101 (192.168.56.101) port 80
> GET /analisis.php?param1=valor1&param2=valor2 HTTP/1.1
> Host: 192.168.56.101
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Wed, 05 Feb 2025 17:49:00 GMT
< Server: Apache/2.4.58 (Ubuntu)
< Vary: Accept-Encoding
< Content-Length: 260
< Content-Type: text/plain; charset=UTF-8
<
Método de la solicitud: GET
URI solicitada: /analisis.php?param1=valor1&param2=valor2

--- Cabeceras de la solicitud ---
Host: 192.168.56.101
User-Agent: curl/8.5.0
Accept: */*

--- Parámetros GET ---
Array
(
    [param1] => valor1
    [param2] => valor2
)
* Connection #0 to host 192.168.56.101 left intact
```

- Las líneas que comienzan con > muestran la solicitud enviada.
- GET /analisis.php?param1=valor1¶m2=valor2 HTTP/1.1: Esta es la línea de solicitud. Muestra el método (GET), la ruta del recurso con los parámetros, y la versión de HTTP.
- Las siguientes líneas son las cabeceras de la solicitud.
- Las líneas que comienzan con < muestran la respuesta del servidor.
- HTTP/1.1 200 OK: Indica que la solicitud fue exitosa.
- Las siguientes líneas son las cabeceras de la respuesta.
- Después de las cabeceras, se ve el cuerpo de la respuesta, que es la salida de nuestro script PHP.

Solicitud POST

Ahora, se realizará la solicitud POST.

```
jarasa03@Jarasa03-Ubuntu-24:~$ curl -v -X POST -d "param1=valor1&param2=valor2" http://192.168.56.101/analisis.php
```

Se va a analizar la salida por trozos para un mayor entendimiento.

```
* Trying 192.168.56.101:80...
* Connected to 192.168.56.101 (192.168.56.101) port 80
> POST /analisis.php HTTP/1.1
> Host: 192.168.56.101
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Length: 27
> Content-Type: application/x-www-form-urlencoded
```

- POST /analisis.php HTTP/1.1: Indica que es una solicitud POST.
- Content-Length: 27: Muestra la longitud del cuerpo de la solicitud.
- Content-Type: application/x-www-form-urlencoded: Indica que los datos se envían como un formulario HTML.

```
< HTTP/1.1 200 OK
< Date: Wed, 05 Feb 2025 18:00:29 GMT
< Server: Apache/2.4.58 (Ubuntu)
< Vary: Accept-Encoding
< Content-Length: 371
< Content-Type: text/plain; charset=UTF-8
<
Método de la solicitud: POST
URI solicitada: /analisis.php

--- Cabeceras de la solicitud ---
Host: 192.168.56.101
User-Agent: curl/8.5.0
Accept: */*
Content-Length: 27
Content-Type: application/x-www-form-urlencoded

--- Parámetros POST ---
Array
(
    [param1] => valor1
    [param2] => valor2
)

--- Cuerpo de la solicitud (RAW data) ---
* Connection #0 to host 192.168.56.101 left intact
```

La respuesta es similar a la de GET, pero ahora vemos los parámetros POST y el cuerpo raw de la solicitud.

Comparación entre GET y POST

1. Método de envío
 - i. GET: Los parámetros se envían a la URL.
 - ii. POST: Los parámetros se envían en el cuerpo de la solicitud.
2. Visibilidad
 - i. GET: Los parámetros son visibles en la URL.
 - ii. POST: Los parámetros no son visibles en la URL.
3. Longitud de datos
 - i. GET: Limitado por la longitud máxima de la URL (varía según el navegador/servidor).
 - ii. POST: Puede enviar una cantidad mayor de datos.
4. Caché
 - i. GET: Las solicitudes pueden ser cacheadas.
 - ii. POST: Las solicitudes no se cachean por defecto.
5. Seguridad
 - i. GET: Menos seguro para datos sensibles debido a su visibilidad.
 - ii. POST: Más seguro para enviar datos sensibles.

Conclusiones

- Este ejercicio permite ver en detalle cómo se procesan diferentes tipos de solicitudes HTTP.
- Se pueden observar las diferencias en cómo se envían y reciben los datos entre GET y POST.
- Esta información es crucial para entender cómo funcionan las aplicaciones web y para implementar medidas de seguridad efectivas.
- El script PHP proporciona una herramienta valiosa para depuración y análisis de solicitudes HTTP.

Parte 2: Implementación de HTTPS con certificados autofirmados

Abrir el puerto 443

Se abre con el siguiente comando.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo ufw allow 443/tcp
Reglas actualizadas
Reglas actualizadas (v6)
```

¿Por qué? Si el puerto 443 no está abierto, se necesita abrirlo en el firewall. HTTPS utiliza TCP, por lo que se especifica TCP.

¿Por qué utilizar un certificado autenticado?

- Es gratuito y fácil de generar.
- Perfecto para entornos de desarrollo y pruebas.
- Permite entender el proceso de configuración de HTTPS sin costos adicionales.

Generar el certificado de autofirmado

¿Por qué? Este comando genera un certificado SSL autofirmado y su clave privada correspondiente. Usamos OpenSSL porque es una herramienta estándar para este propósito. La clave RSA de 2048 bits ofrece un buen equilibrio entre seguridad y rendimiento.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt
```

Explicación de las opciones:

- req: Subcomando para gestionar solicitudes de certificados.
- -x509: Genera un certificado autofirmado en lugar de una solicitud de certificado.
- -nodes: No encripta la clave privada.
- -days 365: Validez del certificado (1 año).
- -newkey rsa:2048: Genera una nueva clave RSA de 2048 bits.
- -keyout: Especifica dónde guardar la clave privada.
- -out: Especifica dónde guardar el certificado.

Ahora se responderán las preguntas de la manera adecuada.

```
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IES La Paloma
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.56.101
Email Address []:javierarruabarrena03@gmail.com
```

- Country Name: Tu código de país (ej. ES para España).
- State or Province Name: Tu provincia.
- Locality Name: Tu ciudad.
- Organization Name: Nombre de tu organización.
- Organizational Unit Name: Departamento (puede dejarse en blanco).
- Common Name: El nombre de dominio de tu servidor o su IP.
- Email Address: Tu dirección de correo electrónico.

Configurar Apache para usar HTTPS

Primero se ha de habilitar el módulo SSL, algo necesario para HTTPS.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo a2enmod ssl
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed certificates.
To activate the new configuration, you need to run:
systemctl restart apache2
```

Ahora se ha de reiniciar Apache para cargar el módulo.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo systemctl restart apache2
```

Ahora se configurará el host virtual SSL, abriendo el archivo de configuración predeterminado para editarlo.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo nano /etc/apache2/sites-available/default-ssl.conf
```

Dentro del mismo, existen las dos siguientes líneas.

```
SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile    /etc/ssl/private/ssl-cert-snakeoil.key
```

Se sustituirán por las siguientes.

```
SSLCertificateFile      /etc/ssl/certs/apache-selfsigned.crt
SSLCertificateKeyFile    /etc/ssl/private/apache-selfsigned.key
```


Esas líneas especifican la ubicación del certificado y de la clave privada.

Ahora se habilitará el sitio SSL.

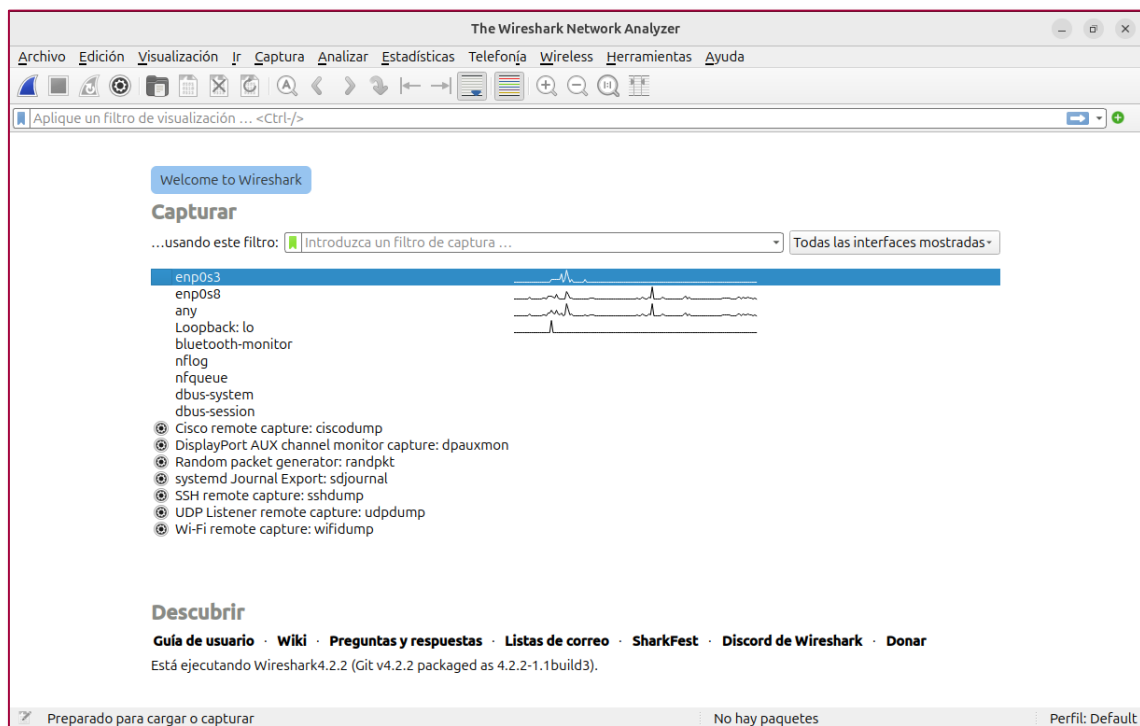
```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo a2ensite default-ssl.conf
Enabling site default-ssl.
To activate the new configuration, you need to run:
systemctl reload apache2
```

Esto activa la configuración SSL que se acaba de modificar. Como bien se indica en la salida del comando anterior, para aplicar los cambios realizados se realizará el siguiente comando.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo systemctl restart apache2
```

Analizar el tráfico con Wireshark

Se abrirá el Wireshark con permisos de administrador, y se seleccionará la tarjeta de red del servidor para analizar el tráfico.



Ahora se harán conexiones primero al http del servidor y luego al https para poder analizar el tráfico.

Análisis del tráfico de http

Cierre de una conexión previa

En el paquete 1 se está cerrando una conexión previa desde la .1 hacia la .101.

En el paquete 2, la máquina .101 responde con un RST, lo que indica que se cerró abruptamente la conexión en lugar de seguir un proceso de cierre normal con FIN, ACK.

1	0.000000000	192.168.56.1	192.168.56.101	TCP	60 51633 → 80 [FIN, ACK] Seq=1 Ack=1 Win=1026 Len=0
2	0.000031096	192.168.56.101	192.168.56.1	TCP	54 80 → 51633 [RST] Seq=1 Win=0 Len=0

Establecimiento de nueva conexión HTTP

Paquete 3: inicia una nueva conexión TCP, con .101 solicitando acceso al puerto 80 (HTTP).

Paquete 4: .1 responde con SYN-ACK, indicando que está listo para aceptar la conexión.

Paquete 5: Parece que .101 reenvía otro SYN, posiblemente una retransmisión.

Paquete 6: .1 responde nuevamente con SYN-ACK, asegurando la sincronización.

Paquete 7: .101 envía el ACK final, completando el handshake de TCP.

3	0.001367517	192.168.56.1	192.168.56.101	TCP	60 51636 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
4	0.001367851	192.168.56.1	192.168.56.101	TCP	60 51637 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
5	0.001394872	192.168.56.101	192.168.56.1	TCP	60 80 → 51636 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
6	0.001537278	192.168.56.101	192.168.56.1	TCP	60 80 → 51637 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	0.001892721	192.168.56.1	192.168.56.101	TCP	60 51636 → 80 [ACK] Seq=1 Ack=1 Win=2097920 Len=0

Petición HTTP

Paquete 9: .101 solicita una página web con GET / HTTP/1.1.

Paquete 10: .1 confirma la recepción del paquete HTTP.

9	0.001892846	192.168.56.1	192.168.56.101	HTTP	615 GET / HTTP/1.1
10	0.001920353	192.168.56.101	192.168.56.1	TCP	54 80 → 51636 [ACK] Seq=1 Ack=562 Win=63744 Len=0

Respuesta del servidor

Paquete 11: El servidor responde con código 200 OK, confirmando que la solicitud fue procesada exitosamente.

Paquete 12: .101 confirma que recibió la respuesta HTTP.

Paquete 13: Un ACK adicional asegurando la recepción de datos.

11	0.003196351	192.168.56.101	192.168.56.1	HTTP	3514 HTTP/1.1 200 OK (text/html)
12	0.003586019	192.168.56.1	192.168.56.101	TCP	60 51636 → 80 [ACK] Seq=562 Ack=2921 Win=2097920 Len=0
13	0.043266721	192.168.56.1	192.168.56.101	TCP	60 51636 → 80 [ACK] Seq=562 Ack=3461 Win=2097408 Len=0

Cierre de conexión

Paquete 14: .101 inicia el cierre de conexión.

Paquete 15: .1 confirma la solicitud de cierre.

14	5.009530917	192.168.56.101	192.168.56.1	TCP	54 80 → 51636 [FIN, ACK] Seq=3461 Ack=562 Win=63744 Len=0
15	5.010359966	192.168.56.1	192.168.56.101	TCP	60 51636 → 80 [ACK] Seq=562 Ack=3462 Win=2097408 Len=0

Análisis del tráfico https

Establecimiento de conexión TCP

Se inicia la conexión TCP entre .101 y .1 en el puerto 443 (HTTPS). Es similar a la conexión HTTP, pero cambia el puerto (80 por 443).

1	0.000000000	192.168.56.1	192.168.56.101	TCP	66 51652 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
2	0.000000228	192.168.56.1	192.168.56.101	TCP	66 51653 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
3	0.000000291	192.168.56.1	192.168.56.101	TCP	66 51654 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
4	0.000030169	192.168.56.101	192.168.56.1	TCP	66 80 → 51652 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.000151110	192.168.56.101	192.168.56.1	TCP	66 443 → 51653 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
6	0.000215291	192.168.56.101	192.168.56.1	TCP	66 443 → 51654 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	0.000464124	192.168.56.1	192.168.56.101	TCP	60 51652 → 80 [ACK] Seq=1 Ack=1 Win=262656 Len=0

Inicio del handshake TLS

Paquete 8: .101 envía un mensaje *Client Hello* al servidor para iniciar el cifrado.

Paquetes 9-10: El servidor responde confirmando la recepción del *Client Hello*.

Paquete 11: El servidor responde con Server Hello, estableciendo los parámetros. Aquí se realiza la negociación del protocolo TLS 1.3.

Paquetes 12-14: Se finaliza la negociación del cifrado y se establece la clave compartida.

8	0.000464165	192.168.56.1	192.168.56.101	TCP	60 51653 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
9	0.000464199	192.168.56.1	192.168.56.101	TCP	60 51654 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
10	0.001011527	192.168.56.1	192.168.56.101	TLSv1.3	1819 Client Hello
11	0.001050099	192.168.56.101	192.168.56.1	TCP	54 443 → 51653 [ACK] Seq=1 Ack=1766 Win=62592 Len=0
12	0.001610566	192.168.56.1	192.168.56.101	TLSv1.3	1851 Client Hello
13	0.001638873	192.168.56.101	192.168.56.1	TCP	54 443 → 51654 [ACK] Seq=1 Ack=1798 Win=62464 Len=0
14	0.005032333	192.168.56.101	192.168.56.1	TLSv1.3	1638 Server Hello, Change Cipher Spec, Application Data, Ap

Intercambio de datos cifrados

Paquetes 15-19: Todos los datos de este bloque están cifrados. No se puede ver el contenido real de la solicitud y la respuesta, a diferencia de HTTP donde era visible GET / HTTP/1.1 y 200 OK.

15	0.005600076	192.168.56.101	192.168.56.1	TLSv1.3	1638 Server Hello, Change Cipher Spec, Application Data, Ap
16	0.005856331	192.168.56.1	192.168.56.101	TCP	60 51654 → 443 [ACK] Seq=1798 Ack=1585 Win=262656 Len=0
17	0.005856541	192.168.56.1	192.168.56.101	TCP	60 51653 → 443 [ACK] Seq=1766 Ack=1585 Win=262656 Len=0
18	0.006613579	192.168.56.1	192.168.56.101	TLSv1.3	84 Change Cipher Spec, Application Data
19	0.006613633	192.168.56.1	192.168.56.101	TLSv1.3	84 Change Cipher Spec, Application Data

Cierre de la conexión

Paquetes 20-22: Se cierra la conexión de forma ordenada, similar a HTTP.

20	0.006613683	192.168.56.1	192.168.56.101	TCP	60 51653 → 443 [FIN, ACK] Seq=1796 Ack=1585 Win=262656 Len=0
21	0.007085779	192.168.56.101	192.168.56.1	TCP	54 443 → 51653 [FIN, ACK] Seq=1585 Ack=1797 Win=62592 Len=0
22	0.007165860	192.168.56.1	192.168.56.101	TCP	60 51654 → 443 [FIN, ACK] Seq=1828 Ack=1585 Win=262656 Len=0

Conclusión

- HTTP envía datos en texto plano, lo que permite ver exactamente qué solicitudes y respuestas se están intercambiando.

- HTTPS protege la comunicación con cifrado TLS, evitando que un atacante pueda ver el contenido de la sesión.
- El tráfico HTTPS solo muestra el handshake inicial y los paquetes cifrados, mientras que en HTTP se pueden leer los métodos (GET), encabezados y respuestas.

Configurar la redirección de HTTP a HTTPS

Primero se editará el archivo de configuración del sitio por defecto.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo nano /etc/apache2/sites-available/000-default.conf
```

Este archivo controla el comportamiento del sitio HTTP por defecto. Se edita para añadir reglas de redirección. Ahora se añadirán las siguientes reglas de escritura. Se añadirán dentro de <VirtualHost *:80>.

```
# Líneas añadidas el 07/02/25 para redirección de http a https
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

- **RewriteEngine On:** Activa el motor de reescritura de Apache.
- **RewriteCond %{HTTPS} off:** Establece una condición para aplicar la regla solo si HTTPS está desactivado.
- **RewriteRule:** Define la regla de redirección.
 - **^(.*)\$:** Captura toda la URL.
 - **https://%{HTTP_HOST}%{REQUEST_URI}:** Redirige a la misma URL pero con https.
 - **[L,R=301]:** L indica que es la última regla, R=301 indica una redirección permanente.

Ahora se habilitará el módulo rewrite de Apache.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo a2enmod rewrite
Enabling module rewrite.
To activate the new configuration, you need to run:
systemctl restart apache2
```

El módulo rewrite es necesario para que las reglas de redirección funcionen. Sin este módulo, Apache ignoraría nuestras reglas. Ahora se reiniciará el servidor de Apache, como bien se indica en la salida del anterior comando.

```
jarasa03@Jarasa03-Ubuntu-24:~$ sudo systemctl restart apache2
```

Esto se realiza para que Apache cargue la nueva configuración y active el módulo rewrite. Ahora al hacer un curl -I a la ip del servidor, se apreciará que este devuelve la Location con https, realizando la redirección con éxito.

```
jarasa03@Jarasa03-Ubuntu-24:~$ curl -I http://192.168.56.101
HTTP/1.1 301 Moved Permanently
Date: Fri, 07 Feb 2025 18:28:17 GMT
Server: Apache/2.4.58 (Ubuntu)
Location: https://192.168.56.101/
Content-Type: text/html; charset=iso-8859-1
```

- 301 Moved Permanently: Indica una redirección permanente, lo que es bueno para SEO.
- Location: Muestra la URL HTTPS a la que se redirige.