

Configuraciones básicas de seguridad para Nginx y Apache

Andrés Guerrero Pinteño
José Carlos Baena Ariza

Introducción

A continuación vamos a mostrar como configurar de una forma muy simple nuestros servidores web Nginx y Apache.

Para Nginx hemos decidido enfocarnos en el control de acceso de HTTP, ocultar la firma y versión del servidor, establecer límites para el Buffer y el tiempo de espera, limitar el número de conexiones por Ip, configurar la seguridad para PHP y, crear y configurar el certificado SSL

Para Apache también hemos ocultado la firma y la versión del servidor, y a parte, hemos deshabilitado el HTTP Trace, configurado el ModSecurity de Apache y hecho una prueba para defendernos de ataques de SQL Injection.

Nginx. Ocultar la versión y firma.

Cuando intentamos entrar a un directorio inexistente, una página no permitida o un error de servidor, tenemos los típicos errores 404, 403 o 500 respectivamente. Como vemos, se muestra la versión del servidor, lo cual puede ser útil para algún atacante, pues puede encontrar vulnerabilidades específicas para la versión en cuestión.



Para solucionar esto, debemos modificar el archivo de configuración de nginx */etc/nginx/nginx.conf* y añadir la siguiente línea, **server_tokens off;**, en la sección *http*. Después de reiniciar el servidor obtenemos lo siguiente.



Nginx. Control de acceso HTTP.

El control de acceso HTTP consiste en implementar un proceso de autenticación para el acceso a algunos archivos y carpetas, esto nos ayuda a proteger los distintos tipos de recursos que disponemos. Es fácil y rápido de implementar y está basado en cabeceras HTTP.

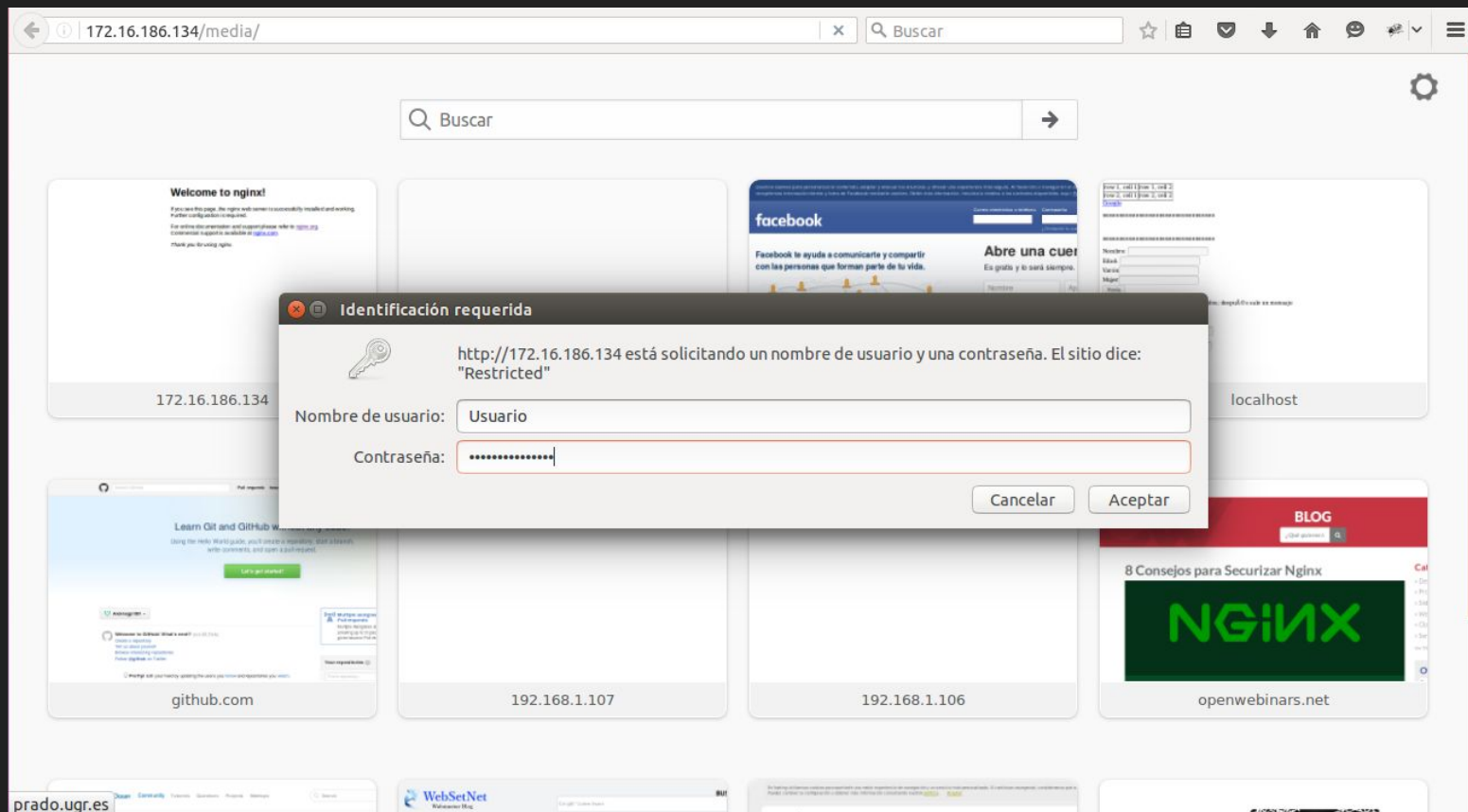
Primero vamos a generar el usuario y la contraseña que deseamos para acceder a la URL, para ello usaremos la librería Apache utils con el comando htpasswd. Este comando nos permite crear un archivo donde almacenaremos el usuario y la contraseña.

```
root@Usuario:/# htpasswd -c /etc/nginx/user_auth Usuario
New password:
Re-type new password:
Adding password for user Usuario
root@Usuario:/#
```

Una vez hecho el archivo con el usuario y la contraseña, tenemos que indicarle a Nginx que debe utilizar dicho archivo para proteger la locación que hayamos decidido, esto agregará una interfaz nueva en el front end del site que protegerá nuestros recursos

```
location /media {
    try_files $uri $uri/ =404;
    auth_basic "Restricted";
    auth_basic_user_file /etc/nginx/user_auth;
}
```

Nginx. Control de acceso HTTP.



Nginx. Establecer límites para Buffer y tiempo de espera

Para evitar ataques de desbordamiento de Buffer creamos el archivo de configuración `/etc/nginx/conf.d/bufer.conf` y limitamos la configuración del Buffer para todos los clientes de la siguiente forma.

```
client_body_buffer_size 1K;
```

```
client_header_buffer_size 1k;
```

```
client_max_body_size 1k;
```

```
large_client_header_buffers 2 1k;
```

También es necesario controlar los tiempos de espera para mejorar el rendimiento del servidor y desconectar clientes.

```
client_body_timeout 10
```

```
client_header_timeout 10;
```

```
keepalive_timeout 5 5;
```

```
send_timeout 10;
```

Nginx. Limitar el número de conexiones Ip.

Para controlar las conexiones de cliente en nginx se utiliza el módulo `ngx_http_limit_zone` que limita el número de conexiones simultáneas. Esta configuración ayuda a evitar ataques de denegación de servicio.

Para esto, haremos uso de la línea con el parámetro `'limit_conn_zone'` y `'limit_conn'`.

```
limit_conn_zone $binary_remote_addr zone=alpha:5m;  
  
server {  
    limit_conn alpha 1;  
    listen 80 default_server;  
    listen [::]:80 default_server;
```

En este caso, hemos establecido que no se podrá hacer más de 5 conexiones por IP.

Para comprobar su funcionamiento, haremos uso del comando `'siege -b -t60S -v http://172.16.186.134/'`

Nginx. Limitar el número de conexiones Ip.

```
Transactions:          66395 hits
Availability:          98.48 %
Elapsed time:          14.79 secs
Data transferred:      24.52 MB
Response time:          0.00 secs
Transaction rate:      4489.18 trans/sec
Throughput:            1.66 MB/sec
Concurrency:           14.91
Successful transactions: 66395
Failed transactions:    1024
Longest transaction:    0.02
Shortest transaction:   0.00
```

FILE: /var/log/siege.log

You can disable this annoying message by editing
the .siegerc file in your home directory; change
the directive 'show-logfile' to false.

[error] unable to create log file: /var/log/siege.log: Permission denied

Nginx. Configuración de seguridad para PHP.

Para poder mejorar la seguridad de nuestro servidor, debemos mejorar también la seguridad de PHP. Para ello vamos a editar el archivo `/etc/php.ini` como sigue.

No permitir las funciones peligrosas

`disable_functions = phpinfo, sistema, correo, Exec`

Tiempo máximo de ejecución de cada script, en segundos

`max_execution_time = 30`

Cantidad máxima de tiempo que cada script puede pasar análisis de petición de datos

`max_input_time = 60`

Cantidad máxima de memoria que puede consumir un script (8MB)

`memory_limit = 8M`

Tamaño máximo de datos puesto que PHP aceptará.

`post_max_size = 8M`

Nginx. Configuración de seguridad para PHP.

Si va a permitir HTTP file uploads.

file_uploads = Apagado

Máximo permitido de tamaño para archivos subidos.

upload_max_filesize = 2M

No exponga los mensajes de error PHP a usuarios externos

display_errors = Apagado

Active el modo seguro

modo seguro = En

Sólo permiten el acceso a archivos ejecutables en el directorio aislado

safe_mode_exec_dir = php-necesaria-ejecutables-trayectoria

Limitar el acceso externo al entorno de PHP

safe_mode_allowed_env_vars = PHP_

Nginx. Configuración de seguridad para PHP.

Restringir la fuga de información de PHP

expose_php = Apagado

Registrar todos los errores

log_errors = En

No se registran globals para entrada de datos

register_globals = Apagado

Minimizar el tamaño admisible de correo PHP

post_max_size = 1K

Asegúrese de que php redirige apropiadamente

cgi.force_redirect = 0

No permitir subir a menos necesario

file_uploads = Apagado

Evite abrir archivos remotos

allow_url_fopen = Apagado

Nginx. Certificado SSL.

Creamos el directorio donde vamos a almacenar la información del SSL y los respectivos archivos de claves y certificado SSL.

```
sudo openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout /etc/nginx/ssl/nginx.key -out /etc/nginx/ssl/nginx.crt
```

```
usuario@Usuario:~$ sudo openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout /etc/nginx/ssl/
nginx.key -out /etc/nginx/ssl/nginx.crt
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to '/etc/nginx/ssl/nginx.key'
-----
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]:ES
State or Province Name (full name) [Some-State]:Granada
Locality Name (eg, city) []:Granada
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UGR
Organizational Unit Name (eg, section) []:ETSIIT
Common Name (e.g. server FQDN or YOUR name) []:dominio.com
Email Address []:admin@dominio.com
.....+++
```

Nginx. Certificado SSL.

A continuación, tenemos que modificar nuestra configuración del archivo **/etc/nginx/sites-enabled/default** en la sección *server* para aprovecharse de ellos ajustando nuestros archivos de bloques de servidor.

```
#           SSL configuration

    listen 443 ssl;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    server_name dominio.com;
```

Nginx. Certificado SSL.

Finalmente, como
siempre, reiniciamos
el servicio para
aplicar los cambios.

```
limit_conn_zone $binary_remote_addr zone=alpha:5m;

server {
    limit_conn alpha 2;
    listen 80 default_server;
    listen [::]:80 default_server;

    #
    SSL configuration

    listen 443 ssl;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    server_name dominio.com;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

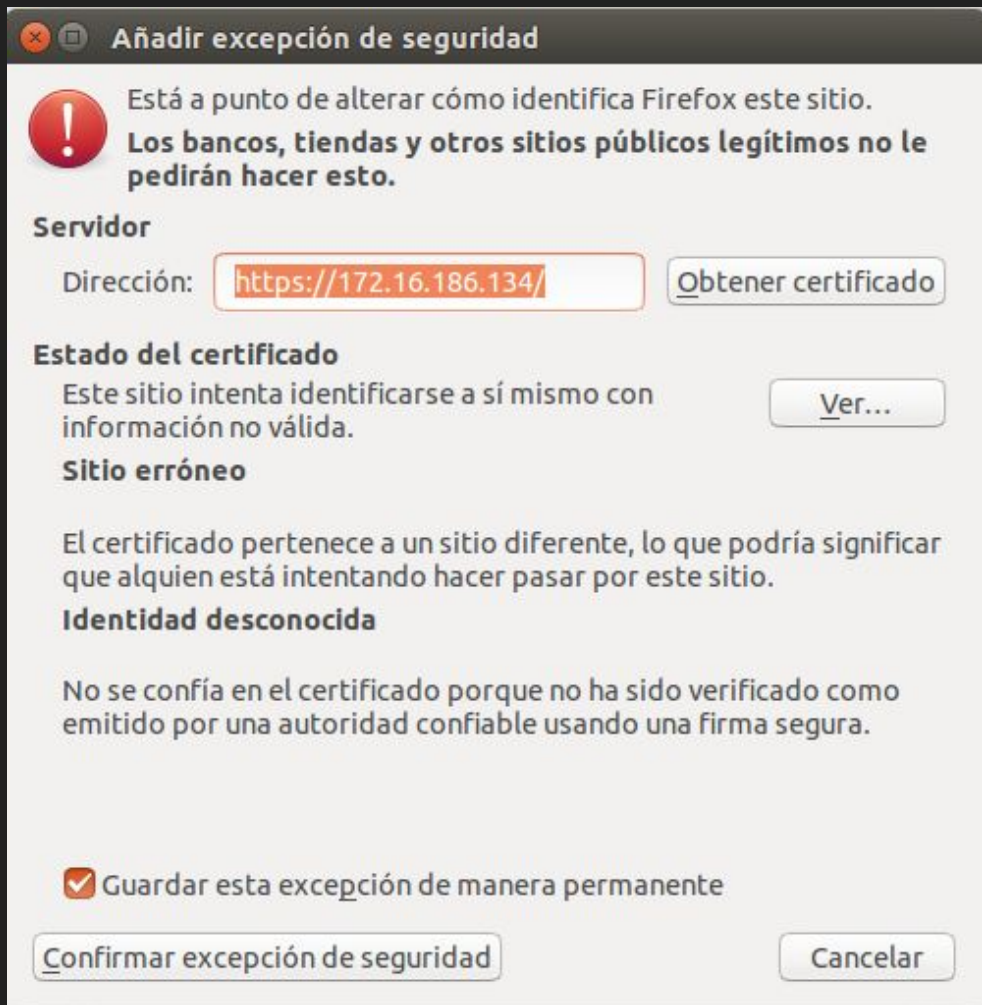
    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;
    }

    location /media {
        try_files $uri $uri/ =404;
        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/user_auth;
    }
}
```

Nginx. Certificado SSL.

Por último, hacemos click en el botón “Avanzado” y nos saldrá un botón para añadir excepción debido a que no tenemos el certificado requerido por el servidor.

Confirmamos la excepción de seguridad para obtener el certificado del sitio web y todo estará listo.



Apache. Ocultar la versión y firma.

Como ya mencionamos en el mismo apartado de nginx, la firma y la versión de nuestro servidor son informaciones valiosas para cualquier atacante, por lo tanto, vamos a realizar la misma práctica de seguridad con apache. Para ello debemos acceder al archivo de configuración **etc/apache2/conf-enabled/security.conf**, buscar la variable 'ServerSignature' y ponerla en off.

```
# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "Email" to also include a mailto: link to the ServerAdmin.
# Set to one of: On | Off | Email
ServerSignature Off
#ServerSignature On
```


Apache. Ocultar la versión y firma.

También debemos cambiar la variable 'ServerTokens' a prod.

```
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of: Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
#ServerTokens Minimal
#ServerTokens OS
#ServerTokens Full
ServerTokens Prod
```

Apache. Deshabilitar HTTP Trace

El HTTP Trace se usa para devolver la información recibida. Puede ser modificado para que devuelva cookies HTTP, dando la posibilidad a que nos roben la sesión HTTP. También puede ser usado para ataques de Cross Site Scripting. Por lo tanto vamos a desactivarlo.

Accedemos al archivo de configuración **etc/apache2/conf-enabled/security.conf** y cambiamos la variable 'TraceEnable' a off.

```
# Allow TRACE method
#
# Set to "extended" to also reflect the request body (only for testing and
# diagnostic purposes).
#
# Set to one of:  On | Off | extended
TraceEnable Off
#TraceEnable On_
```

Apache. Configuración de ModSecurity.

En ModSecurity existen una gran cantidad de reglas que podemos activar para que nuestro servidor apache sea más seguro. Para poder utilizar dichas reglas primero debemos indicarles a apache donde encontrarlas.

Para ello necesitamos modificar el archivo de configuración **etc/apache2/mods-enabled/modsecurity.conf** y añadir las siguientes líneas.

```
<IfModule security2_module>
    # Default Debian dir for modsecurity's persistent data
    SecDataDir /var/cache/modsecurity

    Include "/usr/share/modsecurity-crs/*.conf"
    Include "/usr/share/modsecurity-crs/activated_rules/*.conf"

    # Include all the *.conf files in /etc/modsecurity.
    # Keeping your local configuration in that directory
    # will allow for an easy upgrade of THIS file and
    # make your life easier
    IncludeOptional /etc/modsecurity/*.conf
</IfModule>
```

Apache. Configuración de ModSecurity.

Una vez hecho esto, ya podemos activar las reglas que veamos necesarias para nuestro servidor, en nuestro caso hemos activado las siguientes mediante el comando 'ln -s /usr/share/modsecurity-crs/(carpeta donde se encuentra)/(regla que deseamos)'.

```
root@ubuntu:/usr/share/modsecurity-crs# cd activated_rules/
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ln -s /usr/share/modsecurity-crs/base_rules/modsecurity_crs_23_request_limits.conf
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ls
modsecurity_crs_23_request_limits.conf      README
modsecurity_crs_41_sql_injection_attacks.conf
root@ubuntu:/usr/share/modsecurity-crs# cd activated_rules/
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ln -s /usr/share/modsecurity-crs/experimental_rules/modsecurity_crs_11_brute_force.conf
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ln -s /usr/share/modsecurity-crs/experimental_rules/modsecurity_crs_11_dos_protection.conf
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ln -s /usr/share/modsecurity-crs/experimental_rules/modsecurity_crs_11_slow_dos_protection.conf
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ls
modsecurity_crs_11_brute_force.conf
modsecurity_crs_11_dos_protection.conf
modsecurity_crs_11_slow_dos_protection.conf
modsecurity_crs_23_request_limits.conf
modsecurity_crs_41_sql_injection_attacks.conf
README
```

Apache. Defensa contra SQL Injection.

Para empezar hemos creado un script en php que es un simple formulario con login y contraseña vulnerable a un ataque por SQL Injection.

```
<html>
<body>
    <form action="" method="post">
        Username: <input type="text" name="username"/><br />
        Password: <input type="password" name="password"/><br />
        <input type="submit" name="login" Value="Login"/>
    </form>
<?php
    if(isset($_POST['login']))
    {
        $username = $_POST['username'];
        $password = $_POST['password'];
        $con = mysqli_connect('192.168.1.47','carlos','1234','sample');
        $result = mysqli_query($con, "SELEC * FROM 'users'
            WHERE username='$username' AND password='$password'");
        if(mysqli_num_rows($result) == 0)
            echo 'Invalid username or password';
        else
            echo '<h1>Logged in</h1><p>A secret fo you....</p>';
    }
?>
</body>
</html>
```

Apache. Defensa contra SQL Injection.

A continuación, creamos una base de datos para probar nuestro script. Para ello introducimos el comando 'mysql -u root -p' e introducimos la contraseña del usuario root que pusimos cuando instalamos sql.

```
root@ubuntu:/var/www/html# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
root@ubuntu:/var/www/html#
```

Posteriormente, creamos la base de datos 'sample' y añadimos las tablas.

```
mysql> create database sample;
Query OK, 1 row affected (0.04 sec)

mysql> connect sample;
Connection id: 38
Current database: sample

mysql> create table users(username VARCHAR(100),password VARCHAR(100));
Query OK, 0 rows affected (0.09 sec)

mysql> insert into users values('Carlos','1234');
Query OK, 1 row affected (0.06 sec)

mysql> insert into users values('Andres','0000');
Query OK, 1 row affected (0.06 sec)
```


Apache. Defensa contra SQL Injection.

Una vez creado todo lo relacionado con la base de datos, podemos probar nuestro script, utilizando un login y contraseña que hayamos añadido a la base de datos.



Username:

Password:



Username:

Password:

Logged in

A secret fo you....

A pesar de todo, nuestro script sigue siendo vulnerable a los ataques por SQL Injection, por lo tanto vamos a usar modsecurity para defendernos contra dichos ataques.

Apache. Defensa contra SQL Injection.

Añadimos la regla 'modsecurity_crs_41_sql_injection_attacks.conf' a la carpeta **usr/share/modsecurity-crs/activated_rules**.

```
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ln -s /usr/share/modsecurity-crs/base_rules/modsecurity_crs_41_sql_injection_attacks.conf
root@ubuntu:/usr/share/modsecurity-crs/activated_rules# ls
modsecurity_crs_41_sql_injection_attacks.conf  README
```

Una vez añadida y reiniciado el servicio de apache, podemos probar nuestro ataque por SQL Injection básico.

Username:

Password:

Username:

Password:

Invalid username or password