



UNIVERSIDAD DE GRANADA

P3 - Seguridad (certificados SSL)

Servidores Web de Altas Prestaciones

UGR - ETSIIT

Torres Ramos, Juan Luis

17 abril de 2024

Índice

Tareas Básicas - B1: Preparación del Entorno de Trabajo	1
Tareas Básicas - B5: Docker Compose para la Granja Web con SSL	9
Tareas Básicas - B6: Verificación y Pruebas del Escenario con SSL	12
Tareas Básicas - A1: Exploraciones Avanzadas de creación de certificados SSL	16
Tareas Básicas - A2: Optimización de la configuración SSL en los servidores web	18
Tareas Básicas - A3: Configuración de Caché y Tickets de Sesión SSL en el balanceador	21
Tareas Básicas - A4: Optimización de conexiones HTTPS y cifrado en el balanceador	23

Tareas Básicas - B1: Preparación del Entorno de Trabajo

Vamos a crear nuestro espacio de trabajo, los directorios específicos para los archivos de configuración:

- P3-juanluis-apache para configuraciones de los servidores web.
- P3-juanluis-nginx para configuraciones del balanceador Nginx.
- P3-juanluis-certificados para los certificados autofirmados.

Con un “tree” dentro de la carpeta padre podemos ver los directorios creados

```
tree
.
├── docker-compose.yml
├── Documentacion
│   ├── SWAP_P3_guion.pdf
│   └── SWAP_P3.pdf
├── P3-juanluis-apache
├── P3-juanluis-certificados
├── P3-juanluis-nginx
└── web_juanluis
    └── index.php
```

6 directories, 4 files

Tareas Básicas - B2: Creación de Certificados SSL

Ahora vamos a crear los archivos necesarios para el certificado SSL, tanto la clave privada y certificado autofirmado con OpenSSL en el directorio P3-juanluis-certificados.

Estas son las siguientes especificaciones

1. Generar un certificado: certificado autofirmado
2. Validez de un año
3. Encriptación RSA de 2048 bits
4. Sin necesidad de passphrase
5. Debe ser autofirmado

Para este apartado me he basado en este enlace que es similar a lo pedido. Además explica que hace cada flag para la creación del certificado.

<https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-18-04-es>

Desde la carpeta Practica_3

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout  
./P3-juanluis-certificados/certificado_juanluis.key -out  
./P3-juanluis-certificados/certificado_juanluis.crt
```

- **req:** subcomando para especificar la administración de la solicitud de firma de certificados CSR X.509.
- **x509:** certificado autofirmado
- **nodes:** indica a OPenSSL que omita la opción para proteger nuestro con una frase de contraseña
- **days 365:** establece tiempo certificado válido
- **newkey rsa:2048:** generar nuevo certificado y una nueva clave al mismo tiempo, rsa:2048 indica que cree una clave RSA de 2048 bits de extensión
- **keyout:** donde colocar la llave
- **out:** donde generar el certificado

Cuando metemos el comando tenemos que introducir estos datos:

- **Nombre de país:** ES
- **Provincia:** Granada
- **Localidad:** Granada
- **Organización:** SWAP
- **Organización sección:** Práctica 3
- **Nombre:** juanluis
- **Email:** juanluis-email

Tareas Básicas - B3: Configuración de Servidores Web Apache con SSL

Vamos ahora a adaptar el dockerfile de la práctica 2 para los servidores de Apache para que soporte SSL. Trabajamos en el directorio P3-juanluis-apache. Primero tendremos que crear un archivo de configuración SSL `juanluis-apache-ssl.conf`.

Este archivo incluye

1. Especificamos el puerto 443 para escuchar las peticiones HTTPS
2. Incluimos la ruta certificado SSL y clave privada
3. Las directivas para configurar el host virtual para atender peticiones HTTPS, incluyendo la opción `SSLEngine on`.

juanluis-apache-ssl.conf

```
<VirtualHost *:443>

    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/certificado_juanluis.crt
    SSLCertificateKeyFile /etc/apache2/ssl/certificado_juanluis.key

    SSLProxyEngine On
    SSLProxyCheckPeerCN off
    SSLProxyCheckPeerName off

    <FilesMatch "\.(cgi|shtml|phtml|php)$">
        SSLOptions +StdEnvVars
    </FilesMatch>
    <Directory /usr/lib/cgi-bin>
        SSLOptions +StdEnvVars
    </Directory>

</VirtualHost>
```

Ahora toca modificar el dockerfile Apache de la práctica 1, este debe cumplir

- Hay que instalar los módulos necesarios de Apache para soportar SSL (mod_ssl)
- Activa SSL y crear un directorio para copiar los archivos de certificado y clave privada
- copiar el archivo de configuración ssl anterior en el directorio adecuado /etc/apache2/ssl
- Exponer el puerto 443 (también exponemos el 80), recuerda que en el 443 trabajaremos con https.

DockerFileApacheP3

```
# Establece la imagen base en Debian Buster slim
FROM debian:buster-slim

# Especifica el mantenedor de la imagen
LABEL maintainer="Juan Luis Torres Ramos"

# Actualiza los repositorios de apt-get y luego instala Apache, php y
herramientas de red, finalmente borra los paquetes que acaba de instalar
RUN apt-get update && apt-get install -y \
    apache2 \
    php php-cli \
    iputils-ping \
    netcat \
    traceroute iptables cron \
    && rm -rf /var/lib/apt/lists/*

# Instalar módulo SSL + carpeta para los certificados
RUN a2enmod ssl && mkdir /etc/apache2/ssl

# Configurar el ServerName a localhost y evitar el warning
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf

# Copiar los certificados SSL + configuración de Apache
COPY ./juanluis-apache-ssl.conf
/etc/apache2/sites-available/juanluis-apache-ssl.conf
RUN a2ensite juanluis-apache-ssl.conf

# Exponer el puerto 443 para el tráfico HTTPS
EXPOSE 80 443

# Iniciar Apache por defecto
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Tareas Básicas - B4: Configuración del Balanceador de Carga Nginx con SSL

Ahora vamos a configurar Nginx para balancear la carga entre los servidores Apache configurados con SSL, gestionando las conexiones HTTPS de manera efectiva. Trabajamos desde el directorio P3-juanluis-nginx.

Primero empezamos con el archivo de configuración ssl de nginx. Las especificaciones son las siguientes (adaptamos el archivo de configuración de la práctica 2):

- configurar el servidor para escuchar en el puerto 443 con SSL
- Especificamos la ruta al certificado SSL y a la clave privada

juanluis-nginx-ssl.conf

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    upstream backend_juanluis {
        # round-robin se usa por defecto
        server 192.168.10.9; # web1
        server 192.168.10.2; # web2
        server 192.168.10.3; # web3
        server 192.168.10.4; # web4
        server 192.168.10.5; # web5
        server 192.168.10.6; # web6
        server 192.168.10.7; # web7
        server 192.168.10.8; # web8
    }

    server {
        listen 443 ssl;
        server_name nginx_juanluis;

        access_log /var/log/nginx/nginx_juanluis.access.log;
```

```

error_log /var/log/nginx/nginx_juanluis.error.log;

ssl_certificate /etc/nginx/ssl/certificado_juanluis.crt;
ssl_certificate_key /etc/nginx/ssl/certificado_juanluis.key;

location / {
    proxy_pass http://backend_juanluis;
    proxy_set_header Cookie $http_cookie;
    proxy_hide_header Set-Cookie;
}

location /estadisticas_juanluis {
    stub_status on;
}
}

# crea tmb para el puerto 80
server {
    listen 80;
    server_name nginx_juanluis;

    access_log /var/log/nginx/nginx_juanluis.access.log;
    error_log /var/log/nginx/nginx_juanluis.error.log;

    location / {
        proxy_pass http://backend_juanluis;
        proxy_set_header Cookie $http_cookie;
        proxy_hide_header Set-Cookie;
    }

    location /estadisticas_juanluis {
        stub_status on;
    }
}
}

```


Ahora vamos a modificar el dockerfile para nginx para adaptarlo para manejar el tráfico HTTPS.

Debe de :

- partir de la imagen base oficial
- incluir el archivo de configuración
- exponer puertos 80 y 443
- la parte de copiar los archivos lo hacemos desde el docker compose

DockerFileNginxP3

```
# Partir de la imagen base oficial de Nginx
FROM nginx:latest

# Crear directorio para SSL
RUN mkdir -p /etc/nginx/ssl

# Exponer el puerto 443 para el tráfico HTTPS
EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

Tareas Básicas - B5: Docker Compose para la Granja Web con SSL

Vamos a desarrollar el archivo docker-compose.yml para que despliegue todos los servicios y configuraciones definidos en la práctica.

Cada servicio web debe cumplir

1. imagen construida a partir del dockerFileApacheP3 juanluis-apache-imagen:p3
2. webX donde X es el número de contenedor del 1 al 8
3. volumen a montar web_juanluis en la ruta por defecto de Apache para servir el index.php
4. volumen para montar el directorio local certificados_juanluis en la carpeta /etc/apache2/ssl
5. conexión a las redes red_web y red_servicios con las OP indicadas en el esquema de la práctica, habilita los puertos de cada web

El servicio del balanceador debe cumplir

1. a partir DockerFileNginxP3
2. volumen a montar el archivo juanluis-nginx-ssl.conf en el contenedor /etc/nginx/nginx.conf
3. volumen para montar el directorio local certificados_juanluis en la carpeta /etc/nginx/ssl/.
4. asignacion IP estatica 192.168.10.50
5. dependencia establecida con so servicios de Apache para garantizar el orden correcto despliegue

Ip de los servicios web

- 192.168.10.9 -> web1
- 192.168.10.2 -> web2
- 192.168.10.3 -> web3
- 192.168.10.4 -> web4
- 192.168.10.5 -> web5
- 192.168.10.6 -> web6
- 192.168.10.7 -> web7
- 192.168.10.8 -> web8

EL puerto de balanceador va a ser 300:80 y el 400:443 y para los servicios web corresponderá "8081:80" "9081:443" para web 1, "8082:80" "9082:443" para web 2 y así para el resto.

`docker-compose.yml`

```
version: '3.8'
services:
  # balanceador-nginx:
  balanceador-nginx-ssl:
    build:
      context: ./P3-juanluis-nginx
      dockerfile: DockerFileNginxP3
    image: juanluis-nginx-image:p3
    container_name: balanceador-nginx
    ports:
      - "3000:80"
      - "4000:443"
    volumes:
      - ./P3-juanluis-certificados:/etc/nginx/ssl
      - ./P3-juanluis-nginx/juanluis-nginx-ssl.conf:/etc/nginx/nginx.conf
    networks:
      red_web:
        ipv4_address: 192.168.10.50
    depends_on:
      - web1
      - web2
      - web3
      - web4
      - web5
      - web6
      - web7
      - web8

# servicios web
web1:
  build:
    context: ./P3-juanluis-apache
    dockerfile: DockerFileApacheP3
  image: juanluis-apache-image:p3
  container_name: web1
  ports:
    - "8081:80"
    - "9081:443"
  volumes:
    - ./web_juanluis:/var/www/html
    - ./P3-juanluis-certificados:/etc/apache2/ssl
```

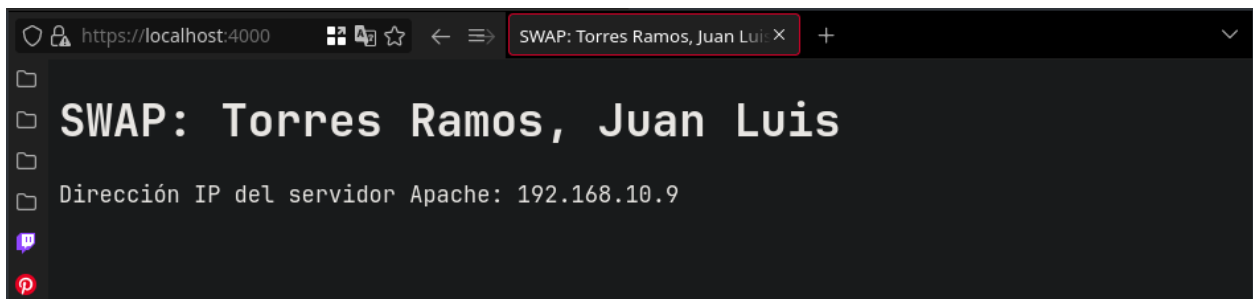
```
-  
./P3-juanluis-apache/juanluis-apache-ssl.conf:/etc/apache2/sites-available/  
juanluis-ssl.conf  
  networks:  
  red_web:  
    ipv4_address: 192.168.10.9  
  red_servicios:  
    ipv4_address: 192.168.20.9  
  
# etc para el resto de web  
networks:  
  red_web:  
    driver: bridge  
    ipam:  
      driver: default  
      config:  
        - subnet: 192.168.10.0/24  
  
  red_servicios:  
    driver: bridge  
    ipam:  
      driver: default  
      config:  
        - subnet: 192.168.20.0/24
```

Tareas Básicas - B6: Verificación y Pruebas del Escenario con SSL

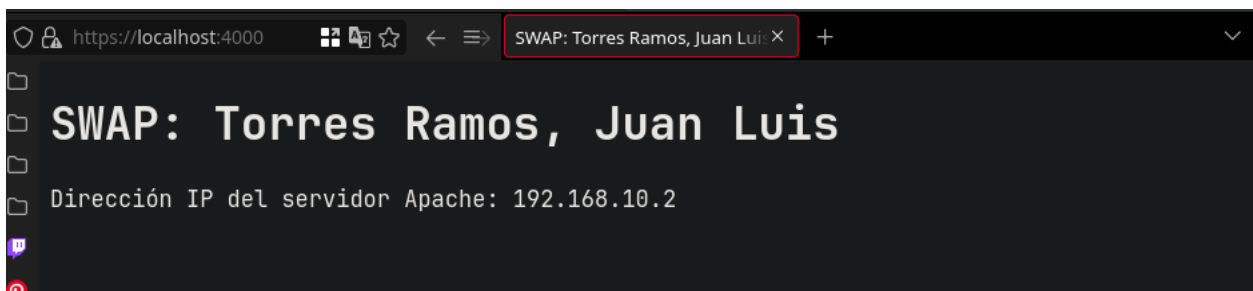
Vamos a comprobar que todo funciona correctamente. Primero construimos y luego desplegamos:

```
docker compose build
docker compose up -d
```

balanceador con el certificado está en <https://localhost:4000/>



si refrescamos irá al siguiente servicio web (web2)



por lo que el balanceador distribuye la carga correctamente (politica round-robin).

Ahora comprobamos que están en funcionamiento el certificado que se ha incluido, podemos ver que en la esquina superior izquierda de la imagen anterior, en la url hay un candado. Ahi si le damos a más información, luego podemos ver la información del certificado que se está usando:

Certificate

juanluis

Subject Name

Country	ES
State/Province	Granada
Locality	Granada
Organization	SWAP
Organizational Unit	Practica 3
Common Name	Juanluis
Email Address	Juanluis

Issuer Name

Country	ES
State/Province	Granada
Locality	Granada
Organization	SWAP
Organizational Unit	Practica 3
Common Name	Juanluis
Email Address	Juanluis

Validity

Not Before	Sun, 05 May 2024 10:06:47 GMT
Not After	Mon, 05 May 2025 10:06:47 GMT

Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	AC:AC:CA:59:CB:CA:E0:9A:E9:73:BB:D1:0E:8A:67:E1:5E:DF:F3:06:10:C5:10:C...

Miscellaneous

Serial Number	33:A7:C9:90:83:D0:06:DF:C8:71:64:5B:42:2C:2E:4A:68:8E:7F:3B
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)

Fingerprints

SHA-256	42:EB:B6:1B:7D:03:2D:42:0B:F5:90:7B:62:75:3E:C0:7B:88:5A:C5:7F:FE:BC:4...
SHA-1	4C:13:58:21:50:5D:BB:E2:45:D6:C8:C7:1B:F4:71:B1:F3:1F:CA:61

Basic Constraints

Certificate Authority	Yes
-----------------------	-----

Subject Key ID

Key ID	E9:A9:F9:6D:8D:DD:F8:98:46:72:B0:F7:1F:ED:A9:E5:16:F1:25:FE
--------	---

Authority Key ID

Key ID	E9:A9:F9:6D:8D:DD:F8:98:46:72:B0:F7:1F:ED:A9:E5:16:F1:25:FE
--------	---

Análisis propuesta IA

Para el apartado básico no he necesitado el uso de la IA. En principio para generar el certificado nos han recomendado que era mejor ver la documentación para entender mejor el funcionamiento de la generación de estos y para el resto de apartados era modificar partes de la práctica 1 y 2 según las especificaciones dadas.

Por otro lado si he tenido un problema a la hora de generar los certificados, porque luego al ejecutar los contenedores el balanceador no tenía permisos para ejecutar los certificados. Este era el error que daba el log del contenedor del balanceador:

```
2024/05/05 11:17:42 [emerg] 1: cannot load certificate key
"/etc/nginx/ssl/certificado_juanluis.key": BIO_new_file() failed (SSL:
error:8000000D:system library::Permission denied:calling
fopen(/etc/nginx/ssl/certificado_juanluis.key, r) error:10080002:BIO
routines::system lib)
```

```
2024-05-05 13:17:42 nginx: [emerg] cannot load certificate key
"/etc/nginx/ssl/certificado_juanluis.key": BIO_new_file() failed (SSL:
error:8000000D:system library::Permission denied:calling
fopen(/etc/nginx/ssl/certificado_juanluis.key, r) error:10080002:BIO
routines::system lib).
```

Para este error intente solucionarlo con Chatgpt, pero no encuentre ninguna solución,, le pase este error y le comente mi situación, esta fue la solución que me dio:

El error indica que Nginx no puede cargar la clave del certificado SSL certificado_juanluis.key debido a un problema de permisos de archivo. El sistema operativo está denegando el acceso al archivo de clave debido a permisos insuficientes.

Para resolver este problema, necesitas ajustar los permisos del archivo certificado_juanluis.key para que Nginx pueda acceder a él. Puedes hacerlo utilizando el comando `chmod` en la terminal:

```
bash
chmod 600 /ruta/al/archivo/certificado_juanluis.key
```

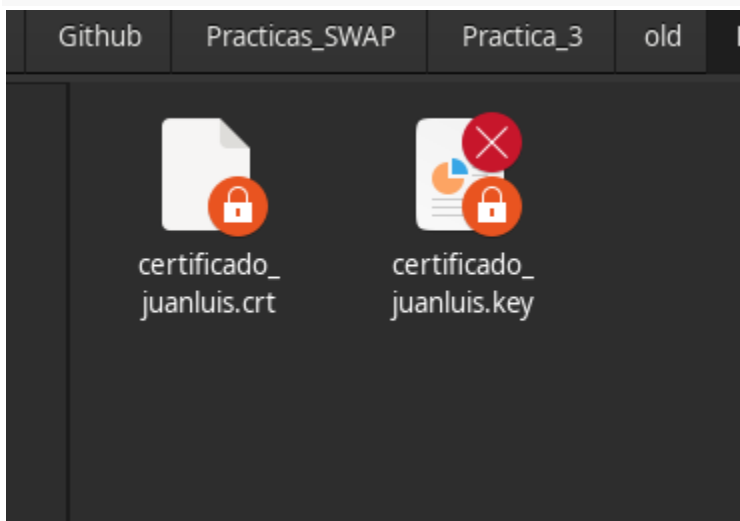
Este comando establecerá los permisos del archivo para que solo el propietario tenga permisos de lectura y escritura, y ningún otro usuario pueda acceder al archivo. Asegúrate de reemplazar `/ruta/al/archivo/certificado_juanluis.key` con la ruta real donde se encuentra tu archivo de clave.

Después de ajustar los permisos del archivo, intenta reiniciar el servicio

de Nginx para que los cambios surtan efecto. Si estás utilizando un contenedor Docker, es posible que necesites volver a construir la imagen del contenedor para aplicar los cambios de permisos en el archivo dentro del contenedor.

De primera instancia dice de cambiar los permisos, pero luego hacía un “ls -l” y eran los permisos correctos, por lo que esta solución no me funcionó, así que acabé recurriendo al maestro. Luego con una solución vi que mis certificados estaban asociados a root por que al hacer openssl hacia tambien “sudo” y los certificados correctos estaban asociados al usuario. Se soluciona con:

```
sudo chown torres:torres certificado_juanluis.crt
sudo chown torres:torres certificado_juanluis.key
```



```
Practicas_SWAP/Practica_3/old/P3-juanluis-certificados □ main [?]
> ls -l
total 8
-rw-r--r-- 1 torres torres 1428 Apr 25 15:31 certificado_juanluis.crt
-rw----- 1 torres torres 1704 Apr 25 15:31 certificado_juanluis.key
```

```
Practicas_SWAP/Practica_3/old/P3-juanluis-certificados-old □ main [?]
> ls -l
total 8
-rw-r--r-- 1 root root 1411 May 5 12:06 certificado_juanluis.crt
-rw----- 1 root root 1704 May 5 12:06 certificado_juanluis.key
```

Por lo general, chat gpt solo recurre a soluciones generales, sobre todo si son errores específicos y no le pasamos los prompt correspondientes, chat gpt no es muy útil, por lo que hay que optar por otras soluciones.

Tareas Básicas - A1: Exploraciones Avanzadas de creación de certificados SSL

Vamos a generar un certificado raíz y certificados intermedios para entender cómo se construye una cadena de confianza. Trabajamos dentro de la carpeta A1 de la carpeta P3-juanluis-certificados.

1. Generamos la clave privada de la CA raíz

```
openssl genrsa -out raizCA.key 4096
```

2. creamos el certificado autofirmado de la raíz

```
openssl req -x509 -new -nodes -key raizCA.key -sha256 -days 365 -out raizCA.crt
```

3. generar clave primaria de la subCA

```
openssl genrsa -out subCA.key 4096
```

4. crear una solicitud de certificado CSR para la subCA

```
openssl req -new -key subCA.key -out subCA.csr
```

el password que pide incluir lo puedes dejar vacío

5. Firmar el CSR de la subCA con la clave privada de la CA raíz

```
openssl x509 -req -in subCA.csr -CA raizCA.crt -CAkey raizCA.key  
-CAcreateserial -out subCA.crt -days 365 -sha256  
Certificate request self-signature ok  
subject=C = ES, ST = Granada, L = granada, O = SWAP, OU = swap, CN =  
juanluis, emailAddress = juanluis
```

Análisis propuesta IA

Generar un certificado raíz (CA) y uno o más certificados intermedios (subCA) para entender cómo se construye una cadena de confianza.

Para generar un certificado raíz (CA) y uno o más certificados intermedios (subCA) para entender cómo se construye una cadena de confianza, sigue estos pasos:

Generar la clave privada de la CA raíz: Utiliza OpenSSL u otra herramienta de generación de claves para crear una clave privada segura. Por ejemplo:

```
csharp
openssl genrsa -out rootCA.key 4096
```

Crear el certificado autofirmado de la CA raíz: Utiliza la clave privada generada para crear un certificado autofirmado que servirá como la CA raíz.

```
vbnet
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 365 -out rootCA.crt
```

Llena la información solicitada, como el nombre de la organización, ubicación, etc.

Generar la clave privada de la subCA: Repite el proceso para generar una clave privada para la subCA:

```
csharp
openssl genrsa -out subCA.key 4096
```

Crear una solicitud de certificado (CSR) para la subCA: Utiliza la clave privada de la subCA para generar una CSR:

```
vbnet
openssl req -new -key subCA.key -out subCA.csr
```

Nuevamente, llena la información solicitada.

Firmar el CSR de la subCA con la clave privada de la CA raíz:

```
objectivec
openssl x509 -req -in subCA.csr -CA rootCA.crt -CAkey rootCA.key
-CACreateserial -out subCA.crt -days 365 -sha256
```

Con estos pasos, has creado una CA raíz y una subCA, y has establecido una cadena de confianza firmada por la CA raíz. Ahora puedes usar la subCA para firmar certificados de servidores, simulando una estructura más realista y segura

Chat gpt me ha dado una solución con los pasos a seguir, no he tenido complicaciones a la hora de seguirla. Muchas veces las respuestas son ambiguas pero en este caso funciona sin problemas, además que te explica cada paso.

Tareas Básicas - A2: Optimización de la configuración SSL en los servidores web

Vamos a optimizar la configuración SSL en Apache para mejorar tanto la seguridad como el rendimiento de las conexiones seguras. Para ello, vamos a deshabilitar protocolos inseguros y cifrados débiles que pueden ser explotados por ataques.

configuración SSL Apache

```
<VirtualHost *:443>

    DocumentRoot /var/www/html

    SSLEngine on

    # Deshabilitar protocolos inseguros
    SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1

    # Deshabilitar cifrados débiles
    SSLCipherSuite HIGH:!MD5:!RC4:!3DES

    SSLCertificateFile /etc/apache2/ssl/certificado_juanluis.crt
    SSLCertificateKeyFile /etc/apache2/ssl/certificado_juanluis.key

    SSLProxyEngine On
    SSLProxyCheckPeerCN off
    SSLProxyCheckPeerName off

    # Opcional: Si deseas configurar un directorio específico para la configuración del
    sitio SSL
    <FilesMatch "\.(cgi|shtml|phtml|php)$">
        SSLOptions +StdEnvVars
    </FilesMatch>
    <Directory /usr/lib/cgi-bin>
        SSLOptions +StdEnvVars
    </Directory>

</VirtualHost>
```

Deshabilitamos los protocolos TLSv1.0 y TLSv1.1 por que tienen vulnerabilidades como POODLE y BEAST (ataques a estos protocolos). Esto se soluciona con los protocolos nuevos TLSv1.2 y TLSv1.3.

El algoritmo MD5 es vulnerable a colisiones y ha sido reemplazado por algoritmos de hash más seguros(SHA-256). El algoritmo RC4 tiene también vulnerabilidades que están relacionadas con el flujo de claves y ataque de cifrado. 3DES es un algoritmo fuerte,pero se caracteriza por ser lento, y se ha acabado sustituyendo por algoritmos como AES)algoritmo de cifrado simétrico más eficiente.

Análisis propuesta IA

adapta el archivo de configuración anterior para optimizarlo deshabilitando protocolos inseguros y deshabilitando los cifrados débiles.

Aquí tienes el archivo de configuración SSL de Apache modificado con las directivas para deshabilitar protocolos y cifrados inseguros:

```
bash
```

```
<VirtualHost *:443>
```

```
    DocumentRoot /var/www/html
```

```
    SSLEngine on
```

```
    # Deshabilitar protocolos inseguros
```

```
    SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1
```

```
    # Deshabilitar cifrados débiles
```

```
    SSLCipherSuite HIGH:!MD5:!RC4:!3DES
```

```
    SSLCertificateFile /etc/apache2/ssl/certificado_juanluis.crt
```

```
    SSLCertificateKeyFile /etc/apache2/ssl/certificado_juanluis.key
```

```
    SSLProxyEngine On
```

```
    SSLProxyCheckPeerCN off
```

```
    SSLProxyCheckPeerName off
```

```
    # Opcional: Si deseas configurar un directorio específico para la configuración del sitio SSL
```

```
    <FilesMatch "\.(cgi|shtml|phtml|php)$">
```

```
        SSLOptions +StdEnvVars
```

```
    </FilesMatch>
```

```
    <Directory /usr/lib/cgi-bin>
```

```
        SSLOptions +StdEnvVars
```

```
    </Directory>
```

```
</VirtualHost>
```

Con estas modificaciones, estás deshabilitando los protocolos TLSv1.0 y TLSv1.1, así como los cifrados MD5, RC4 y 3DES, que se consideran inseguros. Esto mejorará tanto la seguridad como el rendimiento de las conexiones seguras en tu servidor Apache. Recuerda ajustar las rutas de los certificados y claves de acuerdo a tu configuración real.

Chat gpt me ha especificado que cifrados y protocolos he de evitar. Hay que tener cuidado con la información porque el chat gpt no está actualizado con lo último por lo que se recomienda ,corroborar esta información en otras páginas web.

<https://www.comparitech.com/blog/information-security/encryption-types-explained/>

https://httpd.apache.org/docs/trunk/en/ssl/ssl_intro.html

Tareas Básicas - A3: Configuración de Caché y Tickets de Sesión SSL en el balanceador

Vamos a configurar Nginx para utilizar caché de sesiones SSL y tickets de sesión para mejorar la velocidad de las conexiones seguras repetidas, reduciendo el tiempo necesario para negociar la seguridad de la conexión.

Le añadimos lo siguiente dentro de http

```
http {  
    #Para el apartado A2  
    ssl_session_cache shared:SSL:10m; # tamaño de cache y tipo  
    ssl_session_timeout 10m; # tiempo de vida de la cache  
    ssl_session_tickets on; # activar tickets de sesion  
    ...  
}
```

ssl_session_cache: almacena las sesiones SSL en caché para reutilizarse, lo que reduce el tiempo de negociación de la conexión SSL.

ssl_session_timeout: Es el tiempo límite para las sesiones SSL almacenadas en caché.

ssl_session_tickets: Con los tickets se permite que los servidores dejen de almacenar información de sesión SSL. Reduce la carga del servidor y mejora el rendimiento en conexiones HTTPS

Análisis propuesta IA

En este apartado chat gpt me ha enseñado cómo implementar sistema de cache y tickets de sesión para mejorar mi balanceador. Además de explicar su funcionamiento.

Ayúdame a aplicar en mi archivo de configuración nginx, cache de sesiones SSL y tickets de session para mejorar su velocidad

Para configurar NGINX para utilizar caché de sesiones SSL y tickets de sesión y mejorar la velocidad de las conexiones seguras repetidas, puedes seguir estos pasos:

Habilitar la caché de sesiones SSL: Agrega la directiva `ssl_session_cache` en el bloque `http` de tu archivo de configuración de NGINX. Esto almacenará las sesiones SSL en caché para su reutilización, lo que puede reducir el tiempo de negociación de la conexión SSL.

```
nginx
http {
    ssl_session_cache shared:SSL:10m; # Tamaño de La caché y tipo
    ssl_session_timeout 10m; # Tiempo de expiración de La sesión en La caché
    ...
}
```

Habilitar los tickets de sesión SSL: Los tickets de sesión SSL permiten que los servidores dejan de almacenar información de sesión SSL, delegando esta responsabilidad a los clientes. Esto puede reducir la carga del servidor y mejorar el rendimiento en conexiones HTTPS repetidas.

Agrega la directiva `ssl_session_tickets` en el bloque `http` de tu archivo de configuración de NGINX.

```
nginx
http {
    ssl_session_tickets on;
    ...
}
```

Esta configuración habilitará los tickets de sesión SSL en NGINX.

Reinicia NGINX: Una vez que hayas realizado estas modificaciones en tu archivo de configuración, asegúrate de reiniciar NGINX para aplicar los cambios.

```
sudo systemctl restart nginx
```

Con estas configuraciones, NGINX utilizará la caché de sesiones SSL y los tickets de sesión para mejorar la velocidad de las conexiones seguras repetidas, reduciendo así el tiempo necesario para negociar la seguridad de la conexión. Esto puede mejorar significativamente el rendimiento de tu servidor web en el manejo de conexiones HTTPS

Tareas Básicas - A4: Optimización de conexiones HTTPS y cifrado en el balanceador

Vamos a personalizar los protocolos SSL/TLS y suites de cifrado para equilibrar seguridad y rendimiento y activar HTTP/2 para mejorar la eficiencia de las conexiones HTTPS.

Los cifrados que he escogido son ECDH,AESGCM,AES256,EDH. (Son los que me ha recomendado chat gpt)

juanluis-apache-ssl.conf

1. Desactivo los protocolos débiles SSLv3, TLSv1 y TLSv1.1
2. Selecciono los cifrados especificados anteriormente
3. Con SSLHonorCipherOrder on habilito el orden de los cifrados SSL según lo que he especificado en la suite de cifrado

```
...
SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite ECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
SSLHonorCipherOrder on
...
```

juanluis-nginx-ssl.conf

1. Especifico los protocolos TLSv1.2 y TLSv1.3
2. Desactivo la preferencia por los cifrado del servidor, lo que permite que el cliente elija el cifrado.
3. Selecciono los cifrados especificados anteriormente

```
# Para el apartado A4
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off;
ssl_ciphers 'ECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';

# Habilitar HTTP/2 para mejorar la eficiencia de las conexiones HTTPS
http2_max_concurrent_streams 64;

...
server {
    listen 443 ssl http2; # Escuchamos en el puerto 443 con SSL y HTTP/2
    # habilitado.
}
```

Análisis propuesta IA

Ayúdame a personalizar los protocolos SSL/TLS en mis archivos, activa HTTPS2 y que cifrados he de aplicar, dimelo por pasos que cambios tengo que hacer en cada archivo.

Claro, aquí tienes los cambios paso a paso que he realizado en cada archivo:

Archivo de configuración de Nginx (nginx.conf):

Configuración SSL/TLS:

He especificado los protocolos SSL/TLS permitidos utilizando `ssl_protocols TLSv1.2 TLSv1.3`.

Desactivé la preferencia por los cifrados del servidor para permitir que el cliente elija el mejor cifrado disponible usando `ssl_prefer_server_ciphers off`.

Seleccioné suites de cifrado modernas y seguras utilizando `ssl_ciphers 'EECDH:AESGCM:EDH:AESGCM:AES256+EECDH:AES256+EDH'`.

Habilitación de HTTP/2:

Habilité HTTP/2 en el servidor usando `http2_max_concurrent_streams 64`.

Archivo de configuración de Apache (apache.conf):

Configuración SSL/TLS:

Deshabilité los protocolos SSL/TLS inseguros como SSLv3, TLSv1 y TLSv1.1 utilizando `SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1`.

Seleccioné una suite de cifrado sólida utilizando `SSLCipherSuite EECDH:AESGCM:EDH:AESGCM:AES256+EECDH:AES256+EDH`.

Habilité la ordenación de los cifrados SSL para seguir el orden especificado utilizando `SSLHonorCipherOrder on`.

Estos cambios se realizaron para mejorar la seguridad y el rendimiento de las conexiones SSL/TLS en ambos servidores, utilizando protocolos y suites de cifrado modernos y seguros, así como activando HTTP/2 en Nginx para una mejor eficiencia de la comunicación.

Que cifrado me recomiendas para aplicar en los archivos de configuración Para asegurar una combinación óptima de seguridad y rendimiento, te recomendaría utilizar las siguientes suites de cifrado en tus archivos de configuración:

Para Nginx (nginx.conf):

nginx

`ssl_ciphers 'EECDH:AESGCM:EDH:AESGCM:AES256+EECDH:AES256+EDH';`

Para Apache (apache.conf):

apache

SSLCipherSuite ECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH;

Estas suites de cifrado proporcionan una seguridad sólida y un buen rendimiento. Aquí hay un breve resumen de cada componente:

EECDH: Intercambio de claves de curva elíptica, que ofrece seguridad robusta.

AESGCM: Cifrado simétrico utilizando el modo de operación de contador Galois/Counter (GCM), que proporciona seguridad y rendimiento óptimos.

AES256: Cifrado simétrico AES con una clave de 256 bits, que ofrece una seguridad fuerte.

EDH: Intercambio de claves Diffie-Hellman efímero, que añade seguridad adicional.

Estas suites de cifrado están bien equilibradas en términos de seguridad y rendimiento, lo que las hace adecuadas para la mayoría de los casos de uso. Sin embargo, es importante revisar periódicamente las configuraciones de seguridad y actualizarlas según las mejores prácticas recomendadas y las nuevas amenazas que puedan surgir

Siempre hay que intentar entender lo mejor posible cada implementación en nuestro proyecto. Por eso es recomendado tanto corroborar la información como informarse sobre el tema, aquí dejo algunos artículos que he leído para comprender mejor el tema de optimizar nuestro balanceador para que sea más seguro

<https://www.comparitech.com/blog/information-security/beginners-guide-ssl/>

<https://jcweb.es/guia-completa-sobre-como-asegurar-tu-conexion-con-el-protocolo-https/>