



UNIVERSIDAD DE GRANADA

P4 - Seguridad (cortafuegos)

Servidores Web de Altas Prestaciones

UGR - ETSIIT

Torres Ramos, Juan Luis

18 mayo de 2024

Índice

Índice	1
Tareas Básicas - B1: Preparación del Entorno de Trabajo	1
Tareas Básicas - B2: Creación y Configuración de Scripts IPTABLES	3
Tareas Básicas - B3: Implementación de Scripts IPTABLES en Docker	4
Tareas Básicas - B4: Configuración de Docker Compose	6
Tareas Básicas - B5: Verificación y Pruebas	7
Tareas Avanzada - A1: Definir e implementar políticas de seguridad en el balanceador de carga	9
Tareas Avanzada - A2: Configuración Avanzada de IPTABLES para DDoS	11
Análisis propuesta IA	13
Tareas Avanzada - A3: Simular ataques a la granja web y configuraciones de seguridad realizadas	16

Tareas Básicas - B1: Preparación del Entorno de Trabajo

Vamos a establecer el espacio de trabajo de la Práctica 4. Copiamos el directorio creado para la Práctica 3, le cambiamos el nombre a P4-juanluis-X. Dentro de P4-juanluis-apache, creamos un directorio P3-juanluis-iptables-web, donde crearemos los scripts necesarios para IPTABLES para los servidores. La estructura del trabajo quedaría de la siguiente manera:

```
tree
.
├── docker-compose.yml
├── P4-juanluis-apache
│   ├── DockerFileApacheP4
│   ├── juanluis-apache-ssl.conf
│   └── P4-juanluis-iptables-web
├── P4-juanluis-certificados
│   ├── certificado_juanluis.crt
│   └── certificado_juanluis.key
├── P4-juanluis-nginx
│   ├── DockerFileNginxP4
│   └── juanluis-nginx-ssl.conf
└── web_juanluis
    └── index.php
```

6 directories, 8 files

Tareas Básicas - B2: Creación y Configuración de Scripts IPTABLES

Vamos a implementar las políticas de seguridad que vamos a usar en la práctica para nuestros servidores web Apache. El objetivo es denegar el acceso a los servidores desde fuera, para que solo se pueda acceder desde el balanceador. Nos vamos al directorio P4-juanluis-iptables-web y creamos el script juanluis-iptables-web.sh, tiene que incluir dicho script:

1. Denegación implícita de todo el tráfico
2. manejar el tráfico de red entrante basado en el estado de las conexiones
3. manejar el tráfico de red saliente basado en el estado de las conexiones
4. Manejar tráfico de red de la misma máquina
5. manejar tráfico HTTP y HTTPS

juanluis-iptables-web.sh

```
#!/bin/bash

# deneagamos todo el trafico
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# manejamos el trafico de red entrante basado en el estado de Las
conexiones
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# manejamos el trafico de red saliente basado en el estado de Las
conexiones
iptables -A OUTPUT -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT

# manejamos el trafico de red de la misma máquina
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# manejamos el trafico HTTP y HTTPS permitiendo el trafico TCP entrante al
# puerto 80 y 443
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
```

Tareas Básicas - B3: Implementación de Scripts IPTABLES en Docker

Vamos a implementar un `entrypoint.sh`. Un `entrypoint` es una instrucción que especifica el comando que se ejecutará cuando el contenedor se inicie, en nuestro caso será ejecutar `./juanluis-iptables-web.sh`. Para ello tenemos que modificar el `dockerfile` y crear un archivo `entrypoint.sh` dentro de la carpeta `P4-juanluis-iptables-web`:

entrypoint.sh

```
#!/bin/bash
# Ejecuta el script de iptables
./juanluis-iptables-web.sh
# Luego, ejecuta el comando principal del contenedor
exec "$@"
```

luego en el `dockerfile` tendremos que copiar los archivos de `entrypoint.sh` y `./juanluis-iptables-web.sh` al directorio `/`, les damos los permisos adecuados (`chmod +x "archivo"`) a ambos archivos, instalamos `iptables` y añadimos que se ejecute el `entrypoint`:

DockerFileApacheP4

```
# 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 \
    iptables \
    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 script entrada entrypoint y script de reglas iptables

COPY ./P4-juanluis-iptables-web/entrypoint.sh /entrypoint.sh

COPY ./P4-juanluis-iptables-web/juanluis-iptables-web.sh

/juanluis-iptables-web.sh

Darles los permisos de ejecucion

RUN chmod +x /entrypoint.sh /juanluis-iptables-web.sh

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

Configurar el script de entrada

ENTRYPOINT ["/entrypoint.sh"]

Iniciar Apache por defecto

CMD ["apache2ctl", "-D", "FOREGROUND"]

Tareas Básicas - B4: Configuración de Docker Compose

Pasamos a modificar el archivo docker-compose.yml. Partimos de la práctica anterior. Los cambios que se le van a aplicar son

- Los contenedores web tengan las capacidades de red necesarias (cap_add: -NET_ADMIN)

Docker compose

```
...

web1:
  build:
    context: ./P4-juanluis-apache
    dockerfile: DockerFileApacheP4
  image: juanluis-apache-image:p4
  container_name: web1
  ports:
    - "8081:80"
    - "9081:443"
  volumes:
    - ./web_juanluis:/var/www/html
    - ./P4-juanluis-certificados:/etc/apache2/ssl
    -
    ./P4-juanluis-apache/juanluis-apache-ssl.conf:/etc/apache2/sites-available/juanluis-ssl.conf
  cap_add:
    - NET_ADMIN
  networks:
    red_web:
      ipv4_address: 192.168.10.9
    red_servicios:
      ipv4_address: 192.168.20.9

...
```

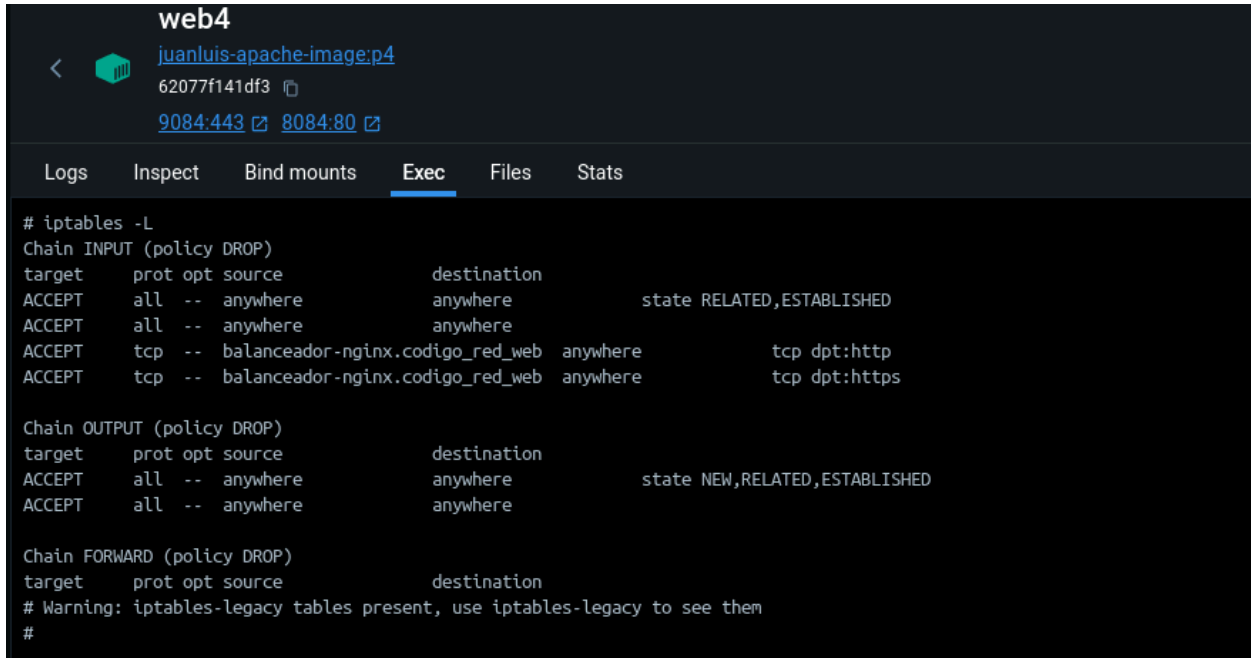
El balanceador se queda igual al de la práctica 3

Tareas Básicas - B5: Verificación y Pruebas

Vamos a comprobar que se han aplicado las reglas iptables en nuestros servicios web

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

entramos desde docker desktop a una de las terminales y listamos las reglas iptables que tiene configuradas:



```
web4
juanluis-apache-image:p4
62077f141df3
9084:443 8084:80

Logs Inspect Bind mounts Exec Files Stats

# iptables -L
Chain INPUT (policy DROP)
target     prot opt source                destination           state RELATED,ESTABLISHED
ACCEPT     all  --  anywhere               anywhere
ACCEPT     all  --  anywhere               anywhere
ACCEPT     tcp  --  balanceador-nginx.codigo_red_web anywhere             tcp dpt:http
ACCEPT     tcp  --  balanceador-nginx.codigo_red_web anywhere             tcp dpt:https

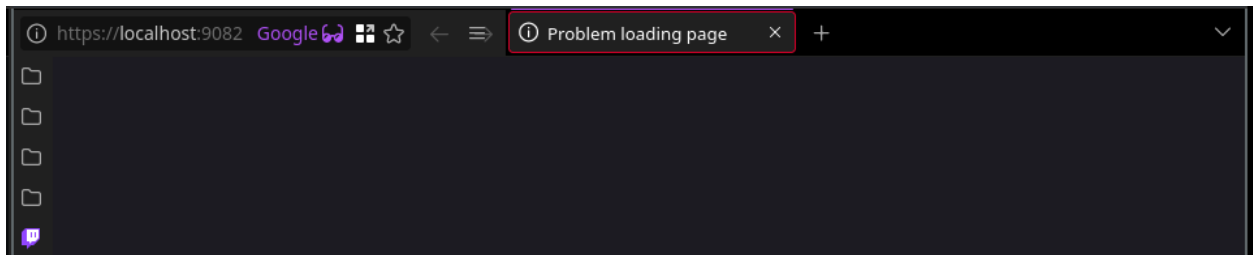
Chain OUTPUT (policy DROP)
target     prot opt source                destination           state NEW,RELATED,ESTABLISHED
ACCEPT     all  --  anywhere               anywhere
ACCEPT     all  --  anywhere               anywhere

Chain FORWARD (policy DROP)
target     prot opt source                destination
# Warning: iptables-legacy tables present, use iptables-legacy to see them
#
```

si comprobamos con el script coinciden las reglas que queríamos aplicar.

Verificamos que Nginx distribuye adecuadamente las solicitudes HTTP y HTTPS y que no podemos acceder directamente a los servidores apache a pesar de que sus puertos están habilitados balanceador: <https://localhost:4000/> <http://localhost:3000/> y para entrar en el servidor apache web2 vamos a usar la dirección <https://localhost:9082/>





para entrar en web2 al final nos da un timeout, por lo que las reglas iptables se están ejecutando correctamente

Tareas Avanzada - A1: Definir e implementar políticas de seguridad en el balanceador de carga

Vamos a definir políticas dentro del balanceador, específicamente dentro de su archivo de configuración. Para este apartado me he basado en un hardening guide en internet, específicamente en el siguiente: <https://www.acunetix.com/blog/web-security-zone/hardening-nginx/>

Políticas implementadas

- uso de SSL/TLS en el puerto 433 para las conexiones HTTPS
- cifrado seguro en la directiva ssl_cipher
- configurado protocolos TLSv1.2 y TLSv1.3
- limitación conexiones HTTP/2 http2_max_concurrent_streams

dentro de “http” he añadido:

```
# 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

# Para el apartado A4
ssl_protocols TLSv1.2 TLSv1.3; # Utilizamos TLSv1.2 y TLSv1.3 para maximizar
la seguridad y el rendimiento.
ssl_prefer_server_ciphers off; # Desactivamos la preferencia por los
cifrados del servidor para permitir que el cliente elija el mejor cifrado
disponible.
ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH'; # Utilizamos
suites de cifrado modernas y seguras para garantizar la confidencialidad y la
integridad de los datos.

# Habilitar HTTP/2 para mejorar la eficiencia de las conexiones HTTPS
http2_max_concurrent_streams 64; # Limitamos el número máximo de corrientes
concurrentes para mantener el rendimiento del servidor.
http2_max_field_size 16k; # Limitamos el tamaño máximo de los campos de
encabezado para evitar ataques de denegación de servicio.
http2_max_header_size 32k; # Limitamos el tamaño máximo de los encabezados
para evitar ataques de denegación de servicio.

server_tokens off; # Desactivamos la información del servidor para evitar la
```

divulgación de información sensible.

client_body_buffer_size 1k; # Limitamos el tamaño del búfer del cuerpo del cliente para evitar ataques de denegación de servicio.

client_header_buffer_size 1k; # Limitamos el tamaño del búfer de encabezado del cliente para evitar ataques de denegación de servicio.

client_max_body_size 1k; # Limitamos el tamaño máximo del cuerpo del cliente para evitar ataques de denegación de servicio.

large_client_header_buffers 2 1k; # Limitamos el tamaño de los búferes de encabezado del cliente para evitar ataques de denegación de servicio.

dentro de “server”, el de https

add_header X-Frame-Options "SAMEORIGIN"; # Evitamos ataques de clickjacking.

add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"; # Habilitamos HSTS para mejorar la seguridad de la conexión.

add_header Content-Security-Policy "default-src 'self' http: https: data: blob: 'unsafe-inline'" always;

add_header X-XSS-Protection "1; mode=block"; # Habilitamos la protección contra ataques de scripting entre sitios.

1. Configuración SSL/TLS y cifrado de la práctica anterior
2. he deshabilitado nginx server_tokens para no mostrar el número de versión de nginx
3. He controlado los recursos y los límites para evitar posibles ataques DoS con directivas como *client_body_buffer_size* o *client_header_buffer_size*
4. Habilitar HTTP/2 para mejorar la eficiencia de las conexiones HTTPS
5. Añadido headers de seguridad como X-Frame-Options para mitigar varios tipos de ataques (clickjacking o ataques de scripting entre sitios)
6. al final del archivo de configuración está implementado el logging de los accesos que se hacen al balanceador (en el archivo *access_log /var/log/nginx/nginx_juanluis.access.log*)
7. También está configurado el monitoreo de estadística básico de prácticas anteriores

Tareas Avanzada - A2: Configuración Avanzada de IPTABLES para DDoS

Ahora vamos a configurar reglas iptables pero en vez en los servidores apache vamos a implementarlo para el balanceador. Seguimos los mismos pasos que con el servidor apache, creamos un script para las reglas iptables y otro para el entryptpoint, copiamos los archivos en el dockerfile, además de descargar iptables, configurar el entryptpoint y darle los permisos correspondiente a los scripts. En el docker consiste en darle permisos de redes al servicio del balanceador.

Vamos a centrarnos en qué reglas vamos a implementar en el balanceador, para para mitigar ataques de Denegación de Servicio Distribuido (DDoS):

- Limitación de la tasa de conexiones nuevas por IP para evitar la saturación de los recursos del servidor.
- Uso de módulos como recent para detectar y bloquear rápidamente el tráfico anómalo y prevenir inundaciones de IPs.
- Configuración de umbrales y reglas específicas que identifiquen patrones de tráfico asociados a ataques comunes de DDoS.
- Protección Contra Ataques de Fragmentación.

Leyendo estos enlaces y basándose en chat gpt he implementado las siguientes reglas

<https://stackoverflow.com/questions/29361694/protecting-nginx-from-dos-attacks>

<https://www.baeldung.com/linux/protection-port-scanners>

```
# 1. Limitación de la tasa de conexiones nuevas por IP para evitar la
# saturación de los recursos del servidor.
iptables -A INPUT -p tcp --syn -m connlimit --connlimit-above 3 -j REJECT
iptables -A INPUT -p tcp --syn -m limit --limit 3/s --limit-burst 6 -j ACCEPT

# 2. Uso de módulos como recent para detectar y bloquear rápidamente el
# tráfico anómalo y prevenir inundaciones de IPs.
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --rcheck --seconds
60 --hitcount 3 -j REJECT
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --remove
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --set -j ACCEPT

iptables -A INPUT -p tcp --dport 443 -m recent --name badguy_https --rcheck
--seconds 60 --hitcount 10 -j REJECT
iptables -A INPUT -p tcp --dport 443 -m recent --name badguy_https --remove
iptables -A INPUT -p tcp --dport 443 -m recent --name badguy_https --set -j
```

ACCEPT

3. Protección Contra Ataques de Fragmentación.

3.1 Bloquear fragmentos de paquetes que no son el primer fragmento

iptables -A INPUT -f -j DROP

3.2 Verificar el tamaño mínimo de fragmentos permitidos

iptables -A INPUT -p tcp -f --fragment -m length --length 40:65535 -j ACCEPT

reglas implementadas

1. La primera regla limita el número de conexiones simultáneas TCP entrantes por IP a 3, rechazando las conexiones adicionales si se supera este límite
2. La segunda regla limita la tasa de conexiones nuevas TCP entrantes a 3 por segundo, con un límite de ráfaga de 6 conexiones. Esto ayuda a prevenir ataques de denegación de servicio basados en la inundación de conexiones TCP
3. Estas reglas utilizan el módulo "recent" para rastrear las conexiones entrantes al puerto 80 (HTTP) y al puerto 443 (HTTPS). Si una IP realiza más de 3 conexiones en un período de 60 segundos en el puerto 80, o más de 10 conexiones en el puerto 443, se realizan las conexiones adicionales. Después de 60 segundos, se eliminará la entrada de la lista de "badguy" (mala IP) y se permitirá el acceso nuevamente si la IP se conecta nuevamente.
4. Las dos últimas reglas se centran en la protección contra ataques de fragmentación

construimos y ejecutamos de nuevo la granja y listamos las reglas iptables del balanceador para ver que se han aplicado correctamente

```
balanceador-nginx
juanluis-nginx-image.p4
65b946f2567b
4000:443 3000:80

STATUS
Running (42 seconds ago)

Logs Inspect Bind mounts Exec Files Stats

# iptables -L
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain INPUT (policy ACCEPT)
target prot opt source destination
REJECT tcp -- anywhere anywhere tcp flags:FIN,SYN,RST,ACK/SYN #conn src/32 > 3 reject-with icmp-port-unreachable
ACCEPT tcp -- anywhere anywhere tcp flags:FIN,SYN,RST,ACK/SYN limit: avg 3/sec burst 6
REJECT tcp -- anywhere anywhere tcp dpt:http recent: CHECK seconds: 60 hit_count: 3 name: badguy side: source mask: 255.255.255.255 reject-with icmp-port-unreachable
tcp -- anywhere anywhere tcp dpt:http recent: REMOVE name: badguy side: source mask: 255.255.255.255
tcp dpt:http recent: SET name: badguy side: source mask: 255.255.255.255
ACCEPT tcp -- anywhere anywhere tcp dpt:https recent: CHECK seconds: 60 hit_count: 10 name: badguy_https side: source mask: 255.255.255.255 reject-with icmp-port-unreachable
REJECT tcp -- anywhere anywhere tcp dpt:https recent: REMOVE name: badguy_https side: source mask: 255.255.255.255
tcp -- anywhere anywhere tcp dpt:https recent: SET name: badguy_https side: source mask: 255.255.255.255
ACCEPT tcp -- anywhere anywhere
DROP all -f anywhere anywhere

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
#
```

Análisis propuesta IA

implementa con iptables las siguientes medidas de seguridad en mi balanceado

Limitación de la tasa de conexiones nuevas por IP para evitar la saturación de los recursos del servidor.

Uso de módulos como recent para detectar y bloquear rápidamente el tráfico anómalo y prevenir inundaciones de IPs.

Configuración de umbrales y reglas específicas que identifiquen patrones de tráfico asociados a ataques comunes de DDoS.

Protección Contra Ataques de Fragmentación

Para implementar todas las reglas solicitadas en iptables para tu balanceador #1, puedes seguir los siguientes pasos. Asumo que tienes privilegios de root y que iptables está instalado y configurado en tu servidor.

Limitación de la tasa de conexiones nuevas por IP para evitar la saturación de los recursos del servidor: Utilizaremos la opción `--limit` para limitar la tasa de nuevas conexiones.

bash

```
iptables -A INPUT -p tcp --syn -m connlimit --connlimit-above 10 -j REJECT
iptables -A INPUT -p tcp --syn -m limit --limit 10/s --limit-burst 20 -j
ACCEPT
```

Estas reglas limitan el número de conexiones TCP nuevas permitidas a 10 por segundo con una ráfaga máxima de 20.

Uso de módulos como recent para detectar y bloquear rápidamente el tráfico anómalo y prevenir inundaciones de IPs: Utilizaremos el módulo recent para rastrear y bloquear IPs sospechosas.

bash

```
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --rcheck
--seconds 60 --hitcount 10 -j REJECT
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --remove
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --set -j ACCEPT
```

Estas reglas bloquearán IPs que intenten más de 10 conexiones en 60 segundos al puerto 80 (HTTP).

Configuración de umbrales y reglas específicas que identifiquen patrones de tráfico asociados a ataques comunes de DDoS: Para esta configuración, utilizaremos varios módulos para identificar patrones comunes de ataques DDoS.

bash

Limitar el número de conexiones simultáneas por IP

```
iptables -A INPUT -p tcp --dport 80 -m connlimit --connlimit-above 20 -j
```

REJECT

Detectar y bloquear IPs con demasiadas conexiones

```
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -m recent --set
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -m recent --update
--seconds 60 --hitcount 30 -j REJECT
```

Estas reglas rechazarán conexiones si una IP tiene más de 20 conexiones simultáneas al puerto 80, o si intenta establecer más de 30 conexiones nuevas en 60 segundos.

Protección Contra Ataques de Fragmentación: Utilizaremos reglas específicas para proteger contra ataques de fragmentación IP.

bash

Bloquear fragmentos de paquetes que no son el primer fragmento

```
iptables -A INPUT -f -j DROP
```

Verificar el tamaño mínimo de fragmentos permitidos

```
iptables -A INPUT -p tcp -f --fragment -m length --length 40:65535 -j
ACCEPT
```

Estas reglas bloquean fragmentos de paquetes (que no sean el primer fragmento) y aseguran que los fragmentos permitidos tengan un tamaño mínimo adecuado.

Implementación Completa

Para facilitar la implementación, puedes poner todas las reglas en un script bash y ejecutarlo. Aquí tienes un ejemplo completo:

bash

```
#!/bin/bash
```

Limitar la tasa de conexiones nuevas por IP

```
iptables -A INPUT -p tcp --syn -m connlimit --connlimit-above 10 -j REJECT
iptables -A INPUT -p tcp --syn -m limit --limit 10/s --limit-burst 20 -j
ACCEPT
```

Detectar y bloquear tráfico anómalo

```
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --rcheck
--seconds 60 --hitcount 10 -j REJECT
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --remove
iptables -A INPUT -p tcp --dport 80 -m recent --name badguy --set -j ACCEPT
```

Configuración de umbrales y reglas específicas

```
iptables -A INPUT -p tcp --dport 80 -m connlimit --connlimit-above 20 -j
REJECT
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -m recent --set
```

```
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -m recent --update  
--seconds 60 --hitcount 30 -j REJECT
```

Protección contra ataques de fragmentación

```
iptables -A INPUT -f -j DROP  
iptables -A INPUT -p tcp -f --fragment -m length --length 40:65535 -j  
ACCEPT
```

Guarda este script en un archivo, por ejemplo iptables_rules.sh, dale permisos de ejecución (chmod +x iptables_rules.sh) y ejecútalo (./iptables_rules.sh). Asegúrate de probar las reglas en un entorno de desarrollo antes de implementarlas en producción para evitar interrumpir el servicio.

Para el apartado A2 me he basado sobre todo en el uso de chat gpt, que me ha generado unas reglas iptable coherentes, que luego he contrastado con algunos enlaces como <https://www.baeldung.com/linux/protection-port-scanners>. Pero luego podemos ver que a la hora de hacer el ataque DDoS (HTTP flood) el servidor se satura, por lo que habría que replantearse estas reglas o implementar algunas nuevas. Por otro lado, he usado chat gpt para comentar mi código y las reglas iptables que he ido implementado para así poder comprenderlo bien.

Por el resto del trabajo, la parte básica solo he tenido que seguir el guión, el apartado A1 me he basado en una guía de “hardening nginx” que me ha sido muy útil y para el apartado 3 he seguido las indicaciones del github para iniciar el ataque.

Tareas Avanzada - A3: Simular ataques a la granja web y configuraciones de seguridad realizadas

Vamos a simular un ataque DDoS con la siguiente botnet llamada Aoyama <https://github.com/Leon123/Aoyama> . Vamos a realizar específicamente un ataque HTTP Flood. Vamos a inundar el balanceador con una gran cantidad de solicitudes HTTP.

1. Lanzamos nuestra granja web

granja web: <https://localhost:4000/> <http://localhost:3000>

2. lanzamos servidor C&C desde la carpeta Aoyama

```
sudo python3 cnc.py 1337
```

y con el siguiente comando nos conectamos al servidor siendo usuario: root , contraseña root

```
telnet 127.0.0.1 1337
contraseña root  usuario: root
```

```
~ 22:49:11
> telnet 127.0.0.1 1337
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused

~ 22:49:13
> telnet 127.0.0.1 1337
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
^
Username:root
Password:root █
```

```
Nodes : 0

d8888
d88888
d88P888
d88P 888 .d88b. 888 888 8888b. 88888b.d88b. 8888b.
d88P 888d88""88b888 888 "88b888 "888 "88b "88b
d88P 888888 888888 888.d888888888 888 888.d888888
d8888888888Y88..88PY88b 888888 888888 888 888888 888
d88P 888 "Y88P" "Y88888"Y888888888 888 888"Y888888
      888
      Y8b d88P
      "Y88P"

root@Aoyama:█
```

3. Vamos a proceder a inicializar a los bots, para ello en otra terminal ejecutamos:
python3 ejecutarBots.py

```
Practica_4/Codigo/P4-juanluis-botnet/Aoyama █ master [?] 22:58:24
> sudo python3 cnc.py 1337
}
Somebody connected:('127.0.0.1', 42562)
Commander here: root
UEBXUQ==
[!] A bot Online ('127.0.0.1', 43968)
UEBXUQ==
[!] A bot Online ('127.0.0.1', 43980)
UEBXUQ==
[!] A bot Online ('127.0.0.1', 43996)
UEBXUQ==
[!] A bot Online ('127.0.0.1', 44002)
UEBXUQ==
[!] A bot Online ('127.0.0.1', 44010)
█

Practica_4/Codigo/P4-juanluis-botnet/Aoyama █ master [?] 22:58:26
o > python3 ejecutarBots.py
Se ha creado el bot 0.
Se ha creado el bot 1.
Se ha creado el bot 2.
Se ha creado el bot 3.
Se ha creado el bot 4.
42029/tcp: 29161
█
```

lanzamos el ataque http flood con el siguiente comando (!http ip port thread ruta)

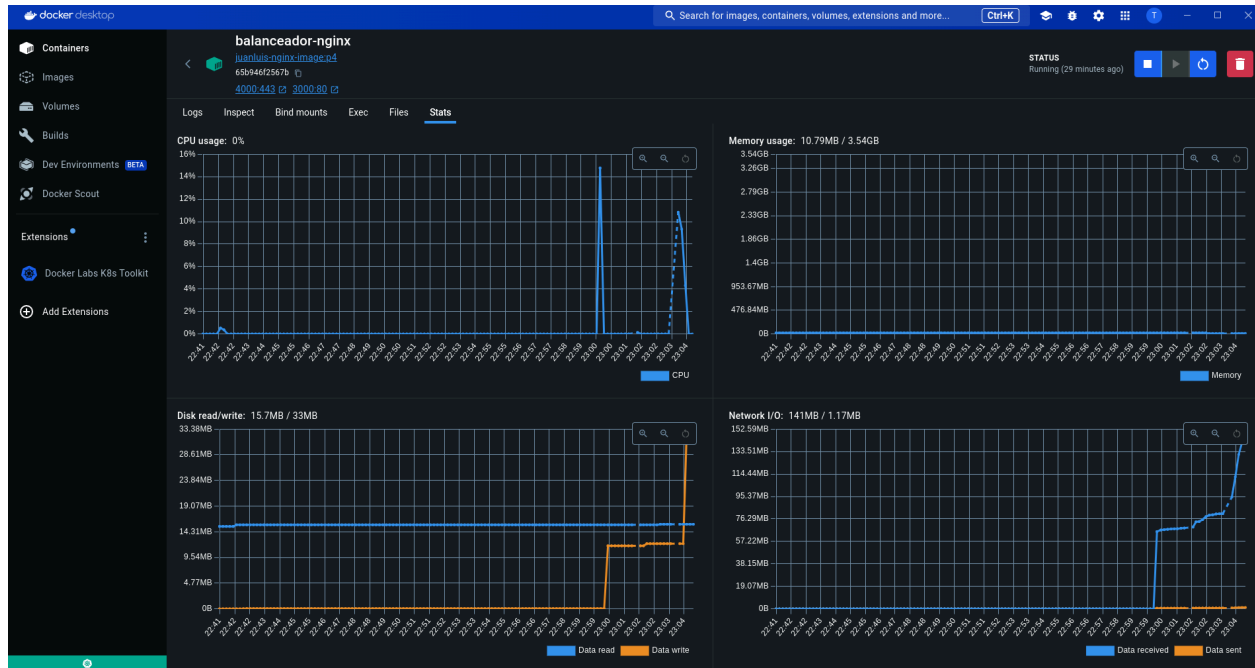
```
!http 127.0.0.1 3000 200 /
```

```
Nodes : 4

d8888
d88888
d88P888
d88P 888 .d88b. 888 888 8888b. 88888b.d88b. 8888b.
d88P 888d88""88b888 888 "88b888 "888 "88b "88b
d88P 888888 888888 888.d888888888 888 888.d888888
d8888888888Y88..88PY88b 888888 888888 888 888888 888
d88P 888 "Y88P" "Y88888"Y888888888 888 888"Y888888
      888
      Y8b d88P
      "Y88P"

root@Aoyama:bots
Nodes:4
root@Aoyama:!http 127.0.0.1 3000 200 /
4 bots exec the command
root@Aoyama:█
```

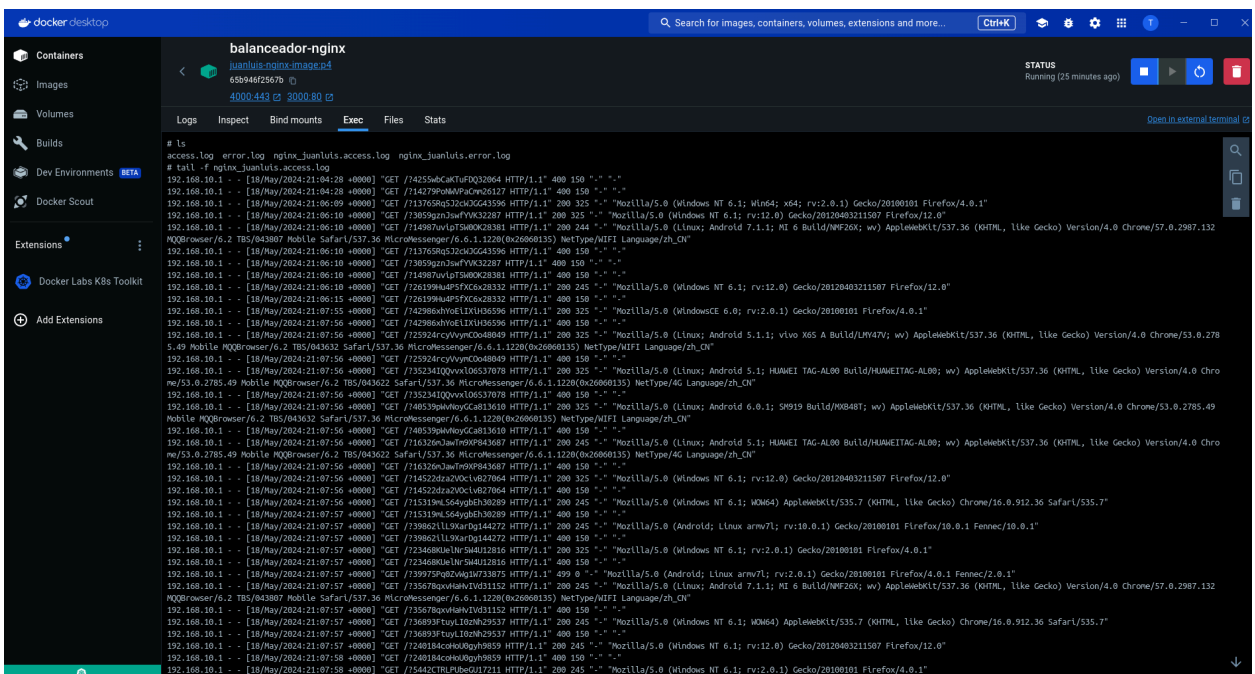
Vemos que pasa dentro del balanceador



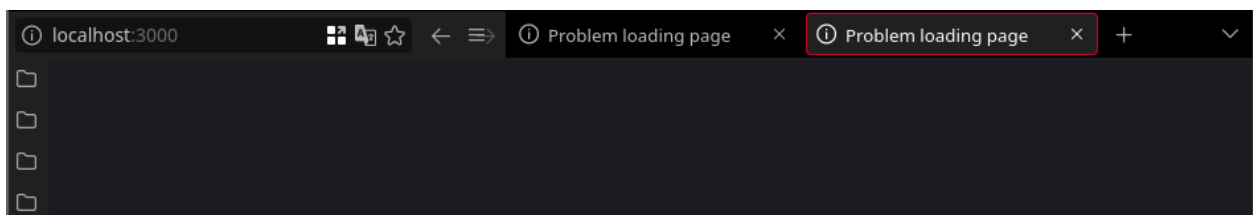
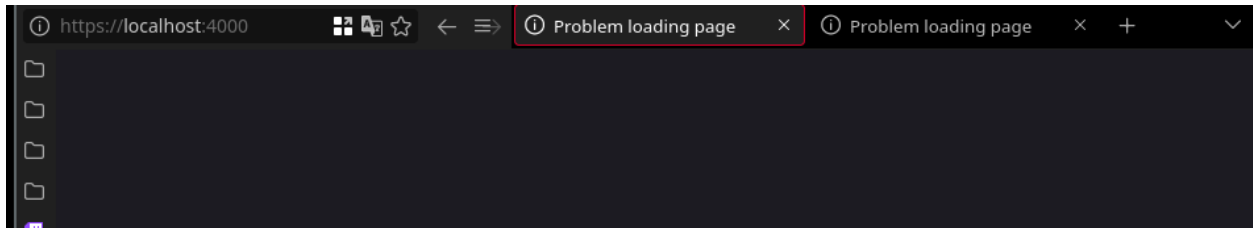
en los stat del balanceador podemos ver que el disco se están escribiendo muchas peticiones, además que en la network están llegando muchas peticiones de llegada.

Previamente en el archivo de configuración de nginx habíamos configurado un archivo de los log del balanceador, dentro del contenedor vamos a /var/log/nginx y hacemos un

```
tail -f nginx_juanluis.access.log
```



Después de iniciar el ataque, notamos que la granja Web está experimentando un gran número de solicitudes GET realizadas por los bots. Si tratamos de recargar la página, podemos notar que la carga es más lenta debido a la gran cantidad de solicitudes, las cuales están causando que el servidor se sature.



Vemos que, a pesar de haber aplicado medidas para mitigar ataques DDoS, con este ataque de 4 bots haciendo un HTTP flood a nuestra granja hace que nuestro servidor se ralentice. Debemos repasar las medidas que hemos implementado e intentar buscar nuevas soluciones. Hay que preguntarle al profesor.