Práctica 4 - Seguridad en Redes

URL del proyecto: https://github.com/PilotoEspacial/SegRed-P4

Contenido

Integrantes

- Alberto Vázquez Martínez - Alberto.Vazquez1@alu.uclm.es
- Paulino de la Fuente Lizcano - Paulino.Lafuente@alu.uclm.es

---

Requisitos

Para poder lanzar el entorno es necesario tener instalado `docker`. Ejecuta los siguientes comandos para instalar `docker` en distribuciones Ubuntu/Debian:

```
sudo apt update
sudo apt-get install docker.io
```

Si quieres evitar escribir `sudo` cada vez que ejecutes el comando `docker`, añade tu nombre de usuario al grupo docker:

```
sudo usermod -aG docker ${USER}
```

Por ultimo es necesario tener instalado **python3** para lanzar las pruebas automaticas disponibles

```
sudo apt-get install python3
```

Setup del entorno

Para automatizar el despliege del entorno, se ha utilizado un archivo `Makefile` con diferentes instrucciones que automatizan las tareas. El órden de ejecución de las instrucciones es el que se muestra a continuación:

1. **build** → construye las imágenes de los diferentes nodos que componen el sistema distribuido.
2. **network** → crea las redes donde se desplegarán los nodos.
3. **containers** → ejecuta los diferentes nodos y les asigna la red a la que pertenecen.
4. **run-tests** → ejecuta tests para comprobar el correcto funcionamiento de la API.
5. **remove** → elimina los contenedores (nodos) y las redes creadas.
6. **clean** → elimina archivos temporales.

Certificados

Para utilizar el protocolo https, es necesario copiar el archivo **broker.cert** a la ruta *usr/local/share/ca-certificates*, una vez ahí, se debe modificar el nombre a **broker.crt** y actualizar la lista de certificados para que la herramienta curl reconozca el certificado

```
$ sudo update-ca-certificates --fresh
```

Modificar archivo para resolución DNS

Según el enunciado, debemos de acceder a la dirección `myserver.local:5000` la cual se traduce en `127.0.0.1:5000`, por ello es necesario modificar el archivo **/etc/hosts** añadiendo una linea con la siguente información

```
172.17.0.2   myserver.local
```

Explicación

Todas estas configuraciones se realizan de manera automática usando `Makefile` y los scripts `entrypoint.sh` de cada contenedor.

Funcionamiento interno de la API

Esta API esta implementada con python y Flask. La forma en la que lo hemos estructurado es la siguiente; Dependiendo del puerto donde se van a atender las peticiones, el router encaminará dichos paquetes a los servidores correspondientes teniendo en cuenta que hay varios servicios corriendo en el sistema, en este caso la **API** utiliza el puerto 5000 para atender las peticiones. En este caso, hemos decidido desglosar la aplicación en 3 partes.

Por un lado tenemos el nodo llamado **broker**, el cual recibirá las peticiones del exterior a traves del router. Este será el encargado de, dependiendo de la petición, llamar al resto de nodos encargados del funcionamiento de la API.

Por otro lado tenemos los servidores **auth** y **files**, están situdado en la misma red.

Auth en concreto se centra en registrar usuarios y asignarles tokens para que estos puedan ejecutar acciones en el servidor **files**. Comprueba si un usuario esta registrado, si su token es valido... relacionado con la seguridad.

Files por el contrario, se centra en crear un espacio para los usuarios donde pueden subir archivos en formato json, de la misma forma pueden editarlos, borrarlos y consultar su contenido.

Iptables para el funcionamiento de la API

Router

```
# Servicios (5000)

iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport 5000 -m state --
state NEW,ESTABLISHED -j ACCEPT
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 5000 -j DNAT --to-
destination 10.0.1.4   #Manda trafico https al nodo broker
iptables -t nat -A POSTROUTING -o eth1 -p tcp --dport 5000 -s 172.17.0.0/16
-d 10.0.1.4 -j SNAT --to-source 10.0.1.2 #Manda trafico https al nodo
broker cambiando la source ip

# Aceptar trafico https entre dmz(eth1) y srv(eth2)

# dmz -> srv
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 5000 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth2 -p tcp --sport 5000 -j ACCEPT

# srv -> dmz
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 5000 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --dport 5000 -j ACCEPT
```

Broker

```
# Servicios (5000)
iptables -A INPUT -p tcp --dport 5000 -i eth0 -s 10.0.1.2 -j ACCEPT  #
Aceptamos trafico del router
iptables -A INPUT -p tcp --sport 5000 -i eth0 -s 10.0.2.4 -j ACCEPT  #
Aceptar trafico https proveniente de files
iptables -A INPUT -p tcp --sport 5000 -i eth0 -s 10.0.2.3 -j ACCEPT  #
Aceptar trafico https proveniente de auth
```

No se adjuntan detalles de los servidores *auth* y *files* ya que es una configuración similar a la del broker, unicamente aceptar los paquetes que provengan de la interfaz eth0, cuyo puerto destino sea el 5000 y por ultimo que el origen sea la ip correspondiente a uno de los dos servidores.

SSH

Para acceder a la máquina work mediante SSH es necesario primero pasar por el nodo jump, ya que no podremos acceder de forma directa al nodo work. Como todo el tráfico debe de pasar por el router, entonces nos conectaremos al router primero y este reenviará el tráfico al nodo jump para conectarnos al nodo work con el usuario que pongamos.

Además, se debe utilizar el usuario jump para poder iniciar el primer salto, por que se establecen una serie de reglas para que se permita la conexión de determinados usuarios. Esta configuración se realiza en el archivo de configuración de ssh del nodo jump.

```
echo -e "Match Address 10.0.1.2\n  AllowUsers jump\n" >>
/etc/ssh/sshd_config
echo -e "Match Address 10.0.3.3\n  AllowUsers op\n" >> /etc/ssh/sshd_config
echo -e "Match Address 10.0.3.3\n  AllowUsers dev\n" >>
/etc/ssh/sshd_config
```

Primero es necesario añadir la clave privada del usuario a nuestro `ssh-agent`, de la siguiente manera:

```
ssh-add docker/assets/ssh/op # Añadir la clave privada del usuario op
ssh-add docker/work/assets/dev # Añadir la clave privada del usuario dev
```

Si nos salta el error: `Could not open a connection to your authentication agent`, será necesario iniciar el ssh-agent:

```
eval `ssh-agent -s`
```

En el caso de que no funcione, puede que sea necesario eliminar del archivo `known_hosts` el host para evitar conflictos:

```
ssh-keygen -f "/home/$USER/.ssh/known_hosts" -R "host_ip_address"
```

Ejecutamos el siguiente comando para conectarnos a al nodo work:

```
ssh -J jump@172.17.0.2 -A op@10.0.3.3
```

Nota: es necesario utilizar la opción -A, ya que necesitaremos la clave privada del usuario con el nos conectamos para poder conectarnos más tarde con ese mismo usuario en diferentes nodos.

Ahora desde el nodo work, podremos acceder a cualquiera de los nodos con el usuario `op` con un simple SSH:

```
op@work:~$ ssh op@10.0.2.3 # Acceder al nodo auth
```

Si intentamos acceder desde el propio host o cualquiera de los demás nodos a al resto de nodos, no se permitirá dicha conexión ya que solo se permiten conexiones SSH provenientes del nodo `work`.

Reglas iptables

Políticas por defecto

En todos los nodos se configurar con las siguientes políticas por defecto:

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

Todo el tráfico entre las diferentes redes debe de ir por el `router`, por lo que se tienen que establecer diferentes reglas FORWARD y NAT para el reenvío y la traducción de la IP origen y destino a las correspondientes.

Ping

Configuración iptables en el nodo router

```
iptables -t nat -A POSTROUTING -o eth0 -p icmp -j MASQUERADE
```

Configuración iptables en todos los nodos

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT
```

SSH

Configuración iptables en el nodo router

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport 22 -m state --
state NEW -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -m state --state ESTABLISHED,RELATED -j
ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -m state --state ESTABLISHED,RELATED -j
ACCEPT
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 22 -j DNAT --to-
destination 10.0.1.3
iptables -t nat -A POSTROUTING -o eth1 -p tcp --dport 22 -s 172.17.0.0/16 -
d 10.0.1.3 -j SNAT --to-source 10.0.1.2

iptables -A FORWARD -i eth1 -o eth3 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth3 -o eth1 -p tcp --sport 22 -j ACCEPT
iptables -A FORWARD -i eth3 -o eth1 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth3 -p tcp --sport 22 -j ACCEPT

iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth2 -p tcp --sport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --dport 22 -j ACCEPT

iptables -A FORWARD -i eth3 -o eth2 -p tcp --sport 22 -j ACCEPT
iptables -A FORWARD -i eth3 -o eth2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth3 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth3 -p tcp --sport 22 -j ACCEPT

iptables -A INPUT -p tcp --dport 22 -i eth3 -s 10.0.3.3 -j ACCEPT
```

Configuración iptables en todos los nodos

```
iptables -A INPUT -p tcp --dport 22 -i eth0 -s 10.0.3.3 -j ACCEPT
```

IP(D)S

Rsyslog

Se ha configurado un nodo específico llamado **logs** que será utilizado para almacenar los logs de todos los nodos del sistema, para así tener un sistema centralizado de logs.

Primero tenemos que configurar el servidor **logs** para usarlo como servidor central. Para ello se utiliza un módulo de `rsyslog` llamado `imupd` que proporciona la capacidad para que el servidor central-rsyslog reciba mensajes Syslog a través del protocolo UDP. Descomentamos las del archivo de configuración **/etc/rsyslog.conf** lineas para activar el módulo:

```
#################
#### MODULES ####
#################
...

# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")

...
```

Además, añadimos una plantilla (50-remote-logs.conf), para especificar la ruta donde se almacenarán los logs de manera centralizada.

En los clientes añadiremos dos plantillas, una para reenviar los logs al servidor de logs y además almacenarlos en manera local (20-forward-logs.conf) y uno específico para el servicio de ssh (50-sshd.conf).

Por último, es necesario aplicar una serie de reglas iptables para el reenvio de las conexiones desde el router. Estas son reglas se aplican en el router para aceptar la conexión al puerto 514 con el protocolo UDP y redirigirlo a la interfaz eth3, que es donde se encuentra el nodo logs.

```
iptables -A FORWARD -i eth1 -o eth3 -p udp --dport 514 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth3 -p udp --dport 514 -j ACCEPT
iptables -A FORWARD -i eth3 -o eth3 -p udp --dport 514 -j ACCEPT

iptables -A INPUT -p udp --dport 514 -i eth3 -s 10.0.3.0/24 -j ACCEPT
iptables -A INPUT -p udp --dport 514 -i eth2 -s 10.0.2.0/24 -j ACCEPT
iptables -A INPUT -p udp --dport 514 -i eth1 -s 10.0.1.0/24 -j ACCEPT
```

Y en el nodo logs, es necesario aceptar el tráfico entrante por ese mismo puerto, de las diferentes redes que forman el sistema.

```
iptables -A INPUT -p udp --dport 514 -i eth0 -s 10.0.1.0/24 -j ACCEPT
iptables -A INPUT -p udp --dport 514 -i eth0 -s 10.0.2.0/24 -j ACCEPT
iptables -A INPUT -p udp --dport 514 -i eth0 -s 10.0.3.0/24 -j ACCEPT
```

Fail2ban

Para evitar ataques de fuerza bruta, se utiliza el servicio de `fail2ban` para banear IPs si exceden un número de intentos a la hora de logear en cualquiera de los nodos mediante SSH. Para ello es necesario tener al menos el un log local monitorize los intentos de autenticación (auth.log). Por eso en la configuración de `rsyslog` se almacenan los logs en un servidor centralizado, pero además, se almacenan de manera local.

La configuración que se realiza en cada nodo es añadir una "jaula" para el servicio de ssh, para que banee una IP que intente autenticarse en cualquiera de los nodos y falle más de 3 veces. El archivo defaults-debian.conf contiene dicha configuración y deberá ser almacenado en la ruta **/etc/fail2ban/jail.d/defaults-debian.conf**.

Hay un excepción en la configuración para el servidor de **logs**, ya que el log que tiene que monitorizar se encuentra en un ruta diferente. Su configuración es la siguiente:

```
[sshd]
enabled = true
port = ssh
mode = aggressive
logpath = /var/log/remotelogs/logs/sshd.log
maxretry = 3
bantime = 120
```

Snort

Configuración

Para la configuración de snort se utilizan los entrypoints de cada uno de los nodos, ya que hay que configurar la interfaz y la red de cada uno. Esta configuración se realiza dentro de la archivo de configuraicón de snort `/etc/snort/snort.debian.conf`.

```
DEBIAN_SNORT_INTERFACE="eth0"
DEBIAN_SNORT_HOME_NET="10.0.X.0/24"
```

La interfaz siempre será la única de cada nodo y la red depende se su ubicación.

Test con traza icmp

Para comprobar el funcionamiento de este sistema de detección de instrusos basado en red, lanzamos un ping con un tamaño mayor al de por defecto. Por ejemplo, utilizo el nodo `files` para lanzar un par de trazas icmp al nodo `auth`.

```
root@files:/# ping -s 900 10.0.2.3
```

Ahora en los logs de snort del nodo auth, vemos como se detecta y se registra una alarma debido a esa incidencia.

```
root@auth:/# cat /var/log/snort/snort.alert.fast
12/28-15:43:18.635782  [**] [1:480:5] ICMP PING speedera [**]
[Classification: Misc activity] [Priority: 3] {ICMP} 10.0.2.4 -> 10.0.2.3
12/28-15:43:18.635782  [**] [1:499:4] ICMP Large ICMP Packet [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {ICMP} 10.0.2.4 ->
10.0.2.3
12/28-15:43:18.635814  [**] [1:499:4] ICMP Large ICMP Packet [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {ICMP} 10.0.2.3 ->
10.0.2.4
12/28-15:43:19.646227  [**] [1:480:5] ICMP PING speedera [**]
[Classification: Misc activity] [Priority: 3] {ICMP} 10.0.2.4 -> 10.0.2.3
12/28-15:43:19.646227  [**] [1:499:4] ICMP Large ICMP Packet [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {ICMP} 10.0.2.4 ->
10.0.2.3
12/28-15:43:19.646259  [**] [1:499:4] ICMP Large ICMP Packet [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {ICMP} 10.0.2.3 ->
10.0.2.4
```

Lanzamiento de pruebas automáticas

Para comprobar el funcionamiento interno de la API, se ha desarrollado un script en python para realizar el testeo, para ello es necesario tener instalado **python3**.

```
make run-tests
```

O si lo prefiere puede ejecutarlo con el siguiente comando

```
python3 ./test
```

Error en la ejecución por modulo Mapping

Si cuando lanzamos el script tenemos errores relacionamos con el modulo *Mapping*, será necesario ejecutar los siguientes comandos para solucionarlo:

```
pip install --upgrade
pip install --upgrade wheel
pip install --upgrade setuptools
pip install --upgrade requests
```

Plano de la red