Práctica 2: Redes

Evolución del software
Grado en Ingeniería Informática

Álvaro Santos Romero

Preparación del entorno (parte 1)

Ejercicio 1.

Para la creación de las máquinas virtuales, vamos a configurar un fichero Vagrantfile para poder lanzarlas simultáneamente.

Mediante la definición de red, le asignamos a cada máquina una ip diferente.

```
Vagrant.configure("2") do |config|
        config.vm.box = "hashicorp/bionic64"

        config.vm.define "vm1" do |vm1|
                vm1.vm.hostname = "vm1"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm1.vm.network "private_network", ip: "192.168.100.10"
        end

        config.vm.define "vm2" do |vm2|
                vm2.vm.hostname = "vm2"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm2.vm.network "private_network", ip: "192.168.100.20"
        end

        config.vm.define "vm3" do |vm3|
                vm3.vm.hostname = "vm3"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm3.vm.network "private_network", ip: "192.168.100.30"
        end
end
```

Mediante este script vamos a provisionar las máquinas con nmap e iptables (por defecto iptables debería de venir instalado).

```
#!/usr/bin/env bash

apt-get install -y nmap
apt-get install -y iptables
```

Una vez lanzadas las máquinas, deberían de tener instaladas estos comandos.

```
vagrant@vm1:~$ nmap --version

Nmap version 7.60 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.3.3 openssl-1.1.0g nmap-libssh2-1.8.0 libz-1.2.8 libpcre-8.39 libpcap-1.8.1 nmap-libdnet-1.12 ip
v6
Compiled without:
Available nsock engines: epoll poll select
vagrant@vm1:~$ iptables --version
iptables v1.6.1
vagrant@vm2:~$ nmap --version

Nmap version 7.60 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.3.3 openssl-1.1.0g nmap-libssh2-1.8.0 libz-1.2.8 libpcre-8.39 libpcap-1.8.1 nmap-libdnet-1.12 ip
v6
Compiled without:
Available nsock engines: epoll poll select
vagrant@vm2:~$ iptables --version
iptables v1.6.1
vagrant@vm3:~$ nmap --version

Nmap version 7.60 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.3.3 openssl-1.1.0g nmap-libssh2-1.8.0 libz-1.2.8 libpcre-8.39 libpcap-1.8.1 nmap-libdnet-1.12 ip
v6
Compiled without:
Available nsock engines: epoll poll select
vagrant@vm3:~$ iptables --version
iptables v1.6.1
```

Configuraciones iptables

Pasos previos:

Añadir a /etc/hosts las diferentes IPs de las máquinas para facilitar los comandos.

Ejercicio 2.

1. Comprobar que puertos tiene abierto VM1 desde VM2.

`nmap --open -p- -v vm1`

```
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
MAC Address: 08:00:27:B5:D6:B9 (Oracle VirtualBox virtual NIC)
```

Como observamos en la captura, la máquina tiene abiertos los puertos **22** y **111**, es decir, son puertos que están siendo utilizados por algún proceso.

Explicación del comando:

- **`-p-`**: Escanea todo el abanico de puertos (1-65535).
- **`-open`**: Muestra sólo puertos abiertos.
- **`-v`**: Verbose, muestra más información sobre el escaneo.

2. Prohibir el acceso por ssh.

`sudo iptables -A INPUT -p tcp --dport 22 -j DROP`

```
vagrant@vm1:~$ sudo iptables -A INPUT -p tcp --dport 22 -j DROP
```

Explicación del comando:

- **`iptables`**: Si no le indicamos una tabla, selecciona la tabla filter por defecto.
- **`-A`**: Añade una regla a la tabla seleccionada.
- **`INPUT`**: Para paquetes de entrada a nuestro sistema.

- **`-p`**: Protocolo afectado por la regla. En este caso **tcp** puesto que **ssh** opera con este.
- **`-dport 22`**: Puerto de entrada afectado por la regla.
- **`-j DROP`**: Indicamos a la tabla qué hacer en cuanto encuentre un objetivo.

`3.` Responder a las siguientes preguntas.

`1.` El acceso a la máquina vía **ssh** se ha perdido, ya que le hemos indicado a el cortafuegos que bloquee toda entrada de conexiones **tcp** por el puerto **22**.

`2.` Si recargamos la máquina, podemos volver a acceder a ella vía **ssh** puesto que las tablas no guardan el estado (habría que utilizar el comando **iptables-persistent**).

`3.` Si, puesto que la configuración de **iptables** se recarga por defecto.

Ejercicio 3.

`1.` Permitir conexiones locales.

```
sudo iptables -A INPUT -p ALL -s localhost -j ACCEPT
```

```
ACCEPT     all  --  localhost            anywhere
```

Explicación del comando:

`-s destino`: para indicar el destino de las conexiones.

`2.` Permitir conexiones ya establecidas.

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED -p ALL -j ACCEPT
```

```
ACCEPT     all  --  anywhere             anywhere             ctstate ESTABLISHED
```

Explicación del comando:

`-m conntrack -ctstate ESTABLISHED`: para indicar que vamos a utilizar el comando ctstate (de la extensión conntrack) para poder filtrar por conexiones ya establecidas.

`3.` Configurar políticas por defecto para rechazo de paquetes.

`sudo iptables -P INPUT DROP`

```
Chain INPUT (policy DROP)
```

Explicación del comando:

- **`-P`**: Añade una política por defecto a una cadena.
- **`INPUT`**: Para paquetes de entrada a nuestro sistema.
- **`-DROP`**: Le indicamos a iptables que la política por defecto va a ser rechazar los paquetes.

Para guardar el entorno configurado, vamos a utilizar el comando **iptables-save > archivo**, que permite exportar las reglas a un archivo, en este caso, en el entorno compartido de vagrant.
De esta forma, podemos importarlas fácilmente desde un archivo con **iptables-restore < archivo**.

```
*filter
:INPUT DROP [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [325:25160]
-A INPUT -s 127.0.0.1/32 -j ACCEPT
-A INPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT
COMMIT
```

Ejercicio 4.

1. Permitir conexión a los puertos **HTTP (80)** y **HTTPS (443)**

```
sudo iptables -A INPUT -p tcp –dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp –dport 443 -j ACCEPT
```

```
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:https
```

Explicación del comando:

- **–p tcp:** Protocolo de la regla **TCP**, ya que **HTTP** y **HTTPS** trabajan en la capa de transporte.
- **–dport:** Puertos destino, en este caso puerto **80** y puerto **443**.
- **-j ACCEPT**: Le indicamos a **iptables** que queremos aceptar las conexiones.

2. Conexión únicamente de **VM2** mediante el servidor **FTP**

```
sudo iptables -A INPUT -p tcp –dport 21 -s vm2 -j ACCEPT
```

```
ACCEPT     tcp  --  vm2                  anywhere             tcp dpt:ftp
```

Explicación del comando:

- **–p tcp:** Protocolo de la regla **TCP**, ya que **FTP** también trabaja en la capa de transporte.
- **–dport:** Puertos destino, en este caso puerto **21**.
- **-s vm2**: Le indicamos a **iptables** que sólo queremos que se conecte **vm2**.
- **-j ACCEPT:** Aceptamos las conexiones.

3. Conexión desde **VM1** a **mysql** únicamente desde **localhost**.

sudo iptables -A INPUT -p tcp –dport 3306 -s localhost -j ACCEPT

```
ACCEPT     tcp  --  localhost            anywhere             tcp dpt:mysql
```

Explicación del comando:

- **–p tcp:** Protocolo de la regla **TCP**, ya que **MYSQL** también trabaja en la capa de transporte.
- **–dport:** Puertos destino, en este caso puerto **3306** que es el puerto por defecto de mysql.
- **-s localhost**: Le indicamos a **iptables** que sólo queremos que se conecte desde localhost.
- **-j ACCEPT:** Aceptamos las conexiones.

Para comprobar que mysql solo aceptará conexiones locales, vamos a intentar conectarnos desde la máquina **VM2** desde el puerto **3306**.

Vamos a ponernos en escucha por el puerto **3306** en **VM1**.

Mediante **telnet** vamos a intentar conectar con **VM1** en ese puerto desde **VM2**.

```
vagrant@vm2:~$ telnet vm1 3306
Trying 192.168.100.10...
telnet: Unable to connect to remote host: Connection timed out
```

Como se observa, no podemos conectarnos.

Si nos conectamos desde la propia máquina, acepta sin problemas.

```
root@vm1:/vagrant# telnet localhost 3306
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hola
que tal
elpepe
```

Ejercicio 5.

Antes de nada, deshabilitamos la conexión desde **VM2** a **VM1** por **ftp (puerto 21)**.

```
sudo iptables -A INPUT -p tcp -s vm2 -dport 21 -j DROP
sudo iptables -A INPUT -p tcp -s vm2 –dport 1:1000 -j ACCEPT
```

```
DROP       tcp  --  vm2                 anywhere            tcp dpt:ftp
ACCEPT     tcp  --  vm2                 anywhere            tcp dpts:tcpmux:1000
```

De esta manera, las conexiones provenientes de VM2 que sean por ftp, serán rechazadas y las demás conexiones entre los puertos 1:1000 se aceptan.

Explicación del comando:

- **`–p tcp:`** Protocolo de la regla **TCP**,ya que al parecer el rango de puertos solo se puede utilizar con el protocolo **tcp**.
- **`–dport 1:1000:`** Puertos destino, en este caso los puertos del 1 al 1000.
- **`-s vm2`**: Le indicamos a **iptables** que sólo queremos que se conecte desde **VM2**.
- **`-j ACCEPT:`** Aceptamos las conexiones.

Ídem para VM3:

```
sudo iptables -A INPUT -p tcp -s vm3 –dport 1:1000 -j ACCEPT
```

```
ACCEPT     tcp  --  vm2                 anywhere            tcp dpts:tcpmux:1000
```

Así es como quedaría la tabla filter de nuestro “servidor http/s”.

```
Chain INPUT (policy DROP)
target     prot opt source              destination
ACCEPT     all  --  localhost           anywhere
ACCEPT     all  --  anywhere            anywhere            ctstate ESTABLISHED
ACCEPT     tcp  --  anywhere            anywhere            tcp dpt:http
ACCEPT     tcp  --  anywhere            anywhere            tcp dpt:https
ACCEPT     tcp  --  vm2                 anywhere            tcp dpt:ftp
ACCEPT     tcp  --  localhost           anywhere            tcp dpt:mysql
DROP       tcp  --  vm2                 anywhere            tcp dpt:ftp
ACCEPT     tcp  --  vm2                 anywhere            tcp dpts:tcpmux:1000
ACCEPT     tcp  --  vm3                 anywhere            tcp dpts:tcpmux:1000
```

Ahora, vamos a comprobar con netcat que las reglas funcionan correctamente.

Antes de nada, para comprobar que **VM2** puede acceder a **VM1** por los puertos **1:1000** menos por **ftp**, deberemos de borrar la regla que permite a **VM2** la conexión a **VM1** por **ftp** (línea 5)

Si nos ponemos en escucha por el puerto **21** (**ftp**), desde **VM2**:

```
vagrant@vm2:~$ telnet 192.168.100.10 21
Trying 192.168.100.10...
telnet: Unable to connect to remote host: Connection timed out
```

Sin embargo, desde **VM3**:

```
vagrant@vm3:~$ telnet 192.168.100.10 21
Trying 192.168.100.10...
Connected to 192.168.100.10.
Escape character is '^]'.
hola
```

Vemos que hay conexión, por lo que la regla está funcionando correctamente.

```
root@vm2:/home/vagrant# telnet 192.168.100.10 1000
Trying 192.168.100.10...
Connected to 192.168.100.10.
Escape character is '^]'.
hola
puerto 1000 que loco
```

```
root@vm3:/home/vagrant# telnet 192.168.100.10 1000
Trying 192.168.100.10...
Connected to 192.168.100.10.
Escape character is '^]'.
tambien hay conexion desde vm3!!
```

Como prueba, se observa que podemos conectarnos desde el puerto **1000** por ejemplo.

Preparación del entorno (parte 2)

Ejercicio 6.

Creación de un entorno de dos subredes y un router.

Dirección ip del router = 192.168.2.1 y 192.168.3.1 (necesitamos tener acceso a las subredes de los equipos)

```
vagrant@router:~$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG        0 0          0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
10.0.2.2        0.0.0.0         255.255.255.255 UH        0 0          0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 eth1
192.168.3.0     0.0.0.0         255.255.255.0   U         0 0          0 eth2
```

Fichero **vagrantfile** para lanzar el sistema:

```
Vagrant.configure("2") do |config|
        config.vm.box = "hashicorp/bionic64"

        config.vm.define "router" do |router|
                router.vm.hostname = "router"
                config.vm.provision :shell, path: "bootstrap.sh"
                router.vm.network "private_network", ip: "192.168.2.1"
                router.vm.network "private_network", ip: "192.168.3.1"

        end

        config.vm.define "nodo1" do |nodo1|
                nodo1.vm.hostname = "nodo1"
                config.vm.provision :shell, path: "bootstrap.sh"
                nodo1.vm.network "private_network", ip: "192.168.2.2"
        end

        config.vm.define "nodo2" do |nodo2|
                nodo2.vm.hostname = "nodo2"
                config.vm.provision :shell, path: "bootstrap.sh"
                nodo2.vm.network "private_network", ip: "192.168.2.3"
        end
         config.vm.define "nodo3" do |nodo3|
                nodo3.vm.hostname = "nodo3"
                config.vm.provision :shell, path: "bootstrap.sh"
                nodo3.vm.network "private_network", ip: "192.168.3.2"
        end
         config.vm.define "nodo4" do |nodo4|
                nodo4.vm.hostname = "nodo4"
                config.vm.provision :shell, path: "bootstrap.sh"
                nodo4.vm.network "private_network", ip: "192.168.3.3"
        end
end
```

Se ha considerado asignar a los equipos una ip diferente a **.1** para evitar conflictos de dispositivo.
Fichero de configuración del Vagrantfile:

```
#!/bin/bash

apt-get install iptables -y
apt-get install traceroute -y
```

Ahora, pasaremos a realizar la configuración del cortafuegos.

- Primero, vamos a habilitar el acceso a las diferentes máquinas mediante ssh cambiando la configuración del fichero **sshd_config.**

  ```
  PasswordAuthentication yes
  ```

Al activar esta opción, permitimos el acceso por ssh utilizando la contraseña de la máquina (es conveniente reiniciar sshd para que se apliquen los cambios).

- A continuación, desde el router nos conectamos a cada nodo mediante ssh y tiramos las interfaces por defecto (en este caso, **eth0**).
  Lo haremos desde el router porque si tiramos la interfaz desde la propia máquina nos quedaremos sin acceso por ssh, ya que nosotros nos conectamos a esta desde esa interfaz con vagrant.

  ```
  vagrant@nodo1:~$ sudo ifconfig eth0
  eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
          inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
          inet6 fe80::a00:27ff:febb:1475  prefixlen 64  scopeid 0x20<link>
          ether 08:00:27:bb:14:75  txqueuelen 1000  (Ethernet)
          RX packets 949  bytes 115056 (115.0 KB)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 693  bytes 116137 (116.1 KB)
          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
  ```

  Cuando tiremos la ruta por defecto, si hacemos un traceroute, veremos que da un error de red inalcanzable, puesto que ahora nuestra máquina no tiene otra ruta hacia internet.

  ```
  vagrant@nodo4:~$ traceroute 8.8.8.8
  traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
  connect: Network is unreachable
  ```

- Una vez hemos tirado las interfaces por defecto de los diferentes equipos, tendremos que añadir la ruta por defecto hacia el router.

```
vagrant@nodo1:~$ sudo ip route add default via 192.168.2.1 dev eth1
```

```
vagrant@nodo1:~$ ip route show
default via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth1 proto kernel scope link src 192.168.2.2
```

Importante: Configurarlo para todas las máquinas.

- Posteriormente, activaremos el **IP-FORWARDING** en el router para retransmitir los paquetes entre redes.
  Para ello, editaremos el fichero **/etc/sysctl.conf** y recargamos las opciones seleccionadas con **sysctl -p.**

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

- Por último, añadiremos una regla a la cadena **IPFORWARDING** de la tabla **nat** para enmascarar la ip origen de los remitentes de un paquete por la de la interfaz pública de una red, ocultando su ip privada.

`sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE`

```
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  0.0.0.0/0            0.0.0.0/0
```

Explicación del comando:

- **`-t nat:`** Indicamos la tabla destino.
- **`-A POSTROUTING:`** Indicamos la cadena destino de la tabla seleccionada.
- **`-o eth0`**: Indicamos la interfaz de salida de la regla, en este caso, la interfaz por defecto del router.
- **`-j MASQUERADE:`** Enmascaramos (sustituimos) la ip del equipo origen de los paquetes, de esta forma, el destinatario del paquete no conocerá la ip del equipo, sino la enmascarada por el router.
  El router mantendrá una tabla de ip´s enmascaradas para posteriormente enviar los paquetes de vuelta al equipo original.

Vamos a comprobar que se ha realizado correctamente:

traceroute desde router a google (8.8.8.8).

```
vagrant@router:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (10.0.2.2)  1.254 ms  0.550 ms  1.357 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
```

Como vemos, sale directamente por la **interfaz pública** a internet.

traceroute desde nodoX (subnet 192.168.2.0/24) a google (8.8.8.8).

```
vagrant@nodo1:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (192.168.2.1)  1.400 ms  0.888 ms  0.837 ms
 2  10.0.2.2 (10.0.2.2)  1.569 ms  3.058 ms  3.696 ms
 3  * * *
 4  * * *
 5  * * *
```

traceroute desde nodoX (subnet 192.168.3.0/24) a google (8.8.8.8).

```
vagrant@nodo3:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (192.168.3.1)  1.330 ms  0.715 ms  1.096 ms
 2  10.0.2.2 (10.0.2.2)  1.584 ms  2.532 ms  3.937 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
```

Sin embargo, en los nodos, primero pasa por la **puerta de enlace** configurada, y posteriormente a la **ip pública** del router.

Ejercicio 7.

Paso 1:
Para realizar esta práctica, vamos a instalar un servidor **DHCP** (**isc-dhcp-server**) en la máquina que actúa como router.

Paso 2/3:
Posteriormente, vamos al fichero de configuración de **dhcp** (**/etc/dhcp/dhcpd.conf**) para configurar las subredes.

```
#DHCP Ejercicio7

subnet 192.168.2.0 netmask 255.255.255.0 {
        #Tiempo de préstamo predeterminado
        default-lease-time 600;
        #Tiempo de préstamo máximo
        max-lease-time 3600;
        #Rango de ips a prestar dentro de la subred
        range 192.168.2.100 192.168.2.200;
}

subnet 192.168.3.0 netmask 255.255.255.0 {
        #Tiempo de préstamo predeterminado
        default-lease-time 600;
        #Tiempo de préstamo máximo
        max-lease-time 3600;
        #Rango de ips a prestar dentro de la subred
        range 192.168.3.100 192.168.3.200;
}
```

Una vez configurado, reiniciamos el servicio (**sudo systemctl isc-dhcp-server**) para cargar la nueva configuración establecida en el fichero de configuración.

A continuación, comprobamos que tenemos el fichero de préstamos (**dhcpd.leases**) en **/var/lib/dhcp/**, si no lo tenemos, lo creamos con **touch**.
Este fichero mantendrá una tabla con los préstamos realizados a los equipos.

Para finalizar, ejecutaremos el comando **sudo dhcpd -f** para ejecutar el servicio en primer plano. Realmente lo usaremos para determinar si todo se ha configurado correctamente.

```
vagrant@nodo1:~$ sudo dhclient -v eth1
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth1/08:00:27:6a:5f:ed
Sending on   LPF/eth1/08:00:27:6a:5f:ed
Sending on   Socket/fallback
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x68057ae0)
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x68057ae0)
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x68057ae0)
DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 3 (xid=0xb2bf0b2d)
DHCPREQUEST of 192.168.2.100 on eth1 to 255.255.255.255 port 67 (xid=0x2d0bbfb2)
DHCPOFFER of 192.168.2.100 from 192.168.2.2
DHCPACK of 192.168.2.100 from 192.168.2.2
bound to 192.168.2.100 -- renewal in 245 seconds.
```

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.100  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::a00:27ff:fe6a:5fed  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6a:5f:ed  txqueuelen 1000  (Ethernet)
        RX packets 247  bytes 43238 (43.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 436  bytes 128518 (128.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Parece que se ha configurado correctamente, puesto que está escuchando en ambas subredes.

Ahora, vamos a pedirle al servidor desde un nodo que nos asigne una IP.

```
vagrant@nodo1:~$ sudo dhclient -v eth1
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth1/08:00:27:6a:5f:ed
Sending on   LPF/eth1/08:00:27:6a:5f:ed
Sending on   Socket/fallback
DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 3 (xid=0xada06856)
DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 8 (xid=0xada06856)
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x5668a0ad)
DHCPOFFER of 192.168.2.5 from 192.168.2.1
DHCPACK of 192.168.2.5 from 192.168.2.1
bound to 192.168.2.5 -- renewal in 287 seconds.
```

Vemos que el servidor nos ha asignado la **IP 192.168.2.100**, la cual está entre el rango

192.168.2.100 - 192.168.2.200, por lo que podemos deducir que funciona correctamente.

En el caso de la otra subred:

```
vagrant@nodo3:~$ sudo dhclient -v eth1
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth1/08:00:27:d5:95:3e
Sending on   LPF/eth1/08:00:27:d5:95:3e
Sending on   Socket/fallback
DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 3 (xid=0x8ea4ae60)
DHCPREQUEST of 192.168.3.100 on eth1 to 255.255.255.255 port 67 (xid=0x60aea48e)
DHCPOFFER of 192.168.3.100 from 192.168.3.1
DHCPACK of 192.168.3.100 from 192.168.3.1
bound to 192.168.3.100 -- renewal in 239 seconds.
```

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.100  netmask 255.255.255.0  broadcast 192.168.3.255
        inet6 fe80::a00:27ff:fed5:953e  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:d5:95:3e  txqueuelen 1000  (Ethernet)
        RX packets 64  bytes 7146 (7.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 38  bytes 5090 (5.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

También vemos que funciona correctamente.

Ahora, vamos a configurar el equipo **host4** de la **subred 2** para que el servidor **dhcp** le asigne una ip estática.
Para ello, vamos a volver al archivo de configuración **dhcpd.conf**.

```
subnet 192.168.3.0 netmask 255.255.255.0 {
        #Tiempo de préstamo predeterminado
        default-lease-time 600;
        #Tiempo de préstamo máximo
        max-lease-time 3600;
        #Rango de ips a prestar dentro de la subred
        range 192.168.3.100 192.168.3.200;

        group{
            #Tiempo de préstamo predeterminado
            default-lease-time 600;
            #Tiempo de préstamo máximo
            max-lease-time 3600;

        host nodo4{

            #Dirección física tarjeta de red por la que mandaremos la petición desde nodo4 (eth1)
            hardware ethernet 08:00:27:32:1c:8b;
            #ip fija del equipo
            fixed-address 192.168.3.69;
            option host-name "nodo4";
            }
        }
}
```

Debemos de crear en el apartado de la subnet 2, un grupo para albergar los diferentes hosts.
En nuestro caso, indicamos que al **nodo4** desde la tarjeta de red (**`eth1:08:00:27:32:1c:b`**) nos asigne la **IP 192.168.3.69**.

NOTA: Importante que la IP que le asignamos no esté dentro del rango de IP´s definido arriba.

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.69  netmask 255.255.255.0  broadcast 192.168.3.255
        inet6 fe80::a00:27ff:fe32:1c8b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:32:1c:8b  txqueuelen 1000  (Ethernet)
        RX packets 225  bytes 29190 (29.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 212  bytes 53904 (53.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Vemos que se le asigna correctamente la IP.

Paso 4:

Estado del fichero dhcpd.leases después de pedir **ip´s** desde los equipos **nodo1**, **nodo2**, **y nodo 3**.

```
lease 192.168.2.103 {
  starts 1 2022/04/11 22:35:55;
  ends 1 2022/04/11 22:45:55;
  cltt 1 2022/04/11 22:35:55;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 08:00:27:6a:5f:ed;
  client-hostname "nodo1";
}
lease 192.168.2.102 {
  starts 1 2022/04/11 22:35:58;
  ends 1 2022/04/11 22:45:58;
  cltt 1 2022/04/11 22:35:58;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 08:00:27:9b:2e:ee;
  client-hostname "nodo2";
}
lease 192.168.3.100 {
  starts 1 2022/04/11 22:35:59;
  ends 1 2022/04/11 22:45:59;
  cltt 1 2022/04/11 22:35:59;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 08:00:27:d5:95:3e;
  client-hostname "nodo3";
}
```

NOTA: En el archivo **dhcpd.leases** no aparece la asignación de IP estáticas, puesto que estas se configuran directamente en la máquina cliente.

Ejercicio 8.

Fichero Vagrantfile para el funcionamiento de la práctica:

```
Vagrant.configure("2") do |config|
        config.vm.box = "hashicorp/bionic64"

        config.vm.define "router" do |router|
                router.vm.hostname = "router"
                config.vm.provision :shell, path: "bootstrapRouter.sh"
                router.vm.network "private_network", ip: "192.168.2.1"
                router.vm.network "private_network", ip: "192.168.3.1"

        end

        config.vm.define "vm1" do |vm1|
                vm1.vm.hostname = "vm1"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm1.vm.network "private_network", ip: "192.168.2.2"
        end

        config.vm.define "vm2" do |vm2|
                vm2.vm.hostname = "vm2"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm2.vm.network "private_network", ip: "192.168.2.3"
        end
         config.vm.define "vm3" do |vm3|
                vm3.vm.hostname = "vm3"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm3.vm.network "private_network", ip: "192.168.3.2"
        end
         config.vm.define "vm4" do |vm4|
                vm4.vm.hostname = "vm4"
                config.vm.provision :shell, path: "bootstrap.sh"
                vm4.vm.network "private_network", ip: "192.168.3.3"
        end
end
```

Fichero de aprovisionamiento del router:

```
#!/bin/bash
apt-get update
apt-get upgrade -y
apt-get install bind9 -y
```

Antes de nada, editaremos el fichero **named.conf.options**.

```
acl "trusted" {
          192.168.2.1;   #dns red 1
          192.168.2.2;   #host 1 red 1
          192.168.2.3;   #host 2 red 1
          192.168.3.1;   #dns red 2
          192.168.3.2;   #host 1 red 2
          192.168.3.3;   #host 2 red 2
        };

options {
        directory "/var/cache/bind";

        recursion yes;
        allow-recursion {trusted; };
        listen-on { 192.168.2.1; 192.168.3.1; };
        allow-transfer {none; };

        forwarders {
          8.8.8.8;
          8.8.4.4;
        };
```

En este fichero, estamos indicando que las ip´s de los hosts y de los servidores están en una categoría llamada "**trusted**", la cuál nos va a permitir que los servidores puedan enviarse las peticiones con la opción **recursion** en caso de no encontrar ese nombre en su servidor (sección **options** del fichero).
También indicamos que sólo escuchen desde las respectivas ip´s **.1**, que en este caso, son los servidores **DNS** de cada **red**.

A continuación, copiamos el fichero **named.defaults-zones** en **named.conf.local** (nos servirá de referencia.)
Añadimos las zonas deseadas.

RED 1:

Zona directa:

```
zone "net1.as.uca.es" {
        type master;
        file "/etc/bind/db.net1.as.uca.es";
};
```

Base de datos de net1.as.uca.es:

```
$TTL    604800
@       IN      SOA     net1.as.uca.es. admin.net1.as.uca.es. (
                              11         ; Serial
                          604800         ; Refresh
                           86400         ; Retry
                         2419200         ; Expire
                          604800 )       ; Negative
;
; Registro de nombre del servidor (red 1)
@       IN      NS      ns1.net1.as.uca.es.

; Registro dirección servidor (red 1)
net1.as.uca.es.         IN      NS      ns1.net1.as.uca.es.

; Registro de los hosts (red 1)
ns1     IN      A       192.168.2.1
vm1     IN      A       192.168.2.2
vm2     IN      A       192.168.2.3
```

Zona inversa:

```
zone "2.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.186.2";
};
```

Base de datos de la zona inversa de net1.as.uca.es:

```
$TTL    604800
@       IN      SOA     ns1.net1.as.uca.es. admin.net1.as.uca.es. (
                              9         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative
;
; Nombre de dominio
@       IN      NS      ns1.
;

1       IN      PTR     ns1.net1.as.uca.es.     ;192.168.2.1; ns1
2       IN      PTR     vm1.net1.as.uca.es.     ;192.168.2.2; vm1
3       IN      PTR     vm2.net1.as.uca.es.     ;192.168.2.3; vm2
```

RED 2:

Zona directa:

```
zone "net2.as.uca.es" {
        type master;
        file "/etc/bind/db.net2.as.uca.es";
};
```

Base de datos de net2.as.uca.es:

Zona inversa:

```
zone "3.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168.3";
};
```

Base de datos de la zona inversa de net2.as.uca.es:

Para comprobar su funcionamiento, al igual que en el **ejercicio 6**, deberemos de configurar las **redes** de tal manera que los hosts tengan conexión entre ellos y que por defecto, pregunten al **router** (esto lo haremos tirando sus interfaces por defecto

y añadiendo nosotros otra que redirija al router mediante las puertas de enlace de este.)

Una vez realizada la configuración, podremos empezar a configurar el fichero **/etc/resolv.conf** para añadir los servidores de nombre del router.

```
nameserver 127.0.0.53
nameserver 192.168.2.1
nameserver 192.168.3.1
options edns0
```

De esta forma, al realizar peticiones **DNS**, utilizará los servidores del router en caso de no encontrar la información en el por defecto.

Desde los hosts de la red 1 :

Nota: Sólo se adjuntarán capturas del primer host, puesto que el segundo es idéntico y por lo tanto, sería información redundante.

Resolución directa:

```
vagrant@vm1:~$ nslookup vm1.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.2.1
Address:        192.168.2.1#53

Name:   vm1.net1.as.uca.es
Address: 192.168.2.2
```

```
vagrant@vm1:~$ nslookup vm2.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.2.1
Address:        192.168.2.1#53

Name:   vm2.net1.as.uca.es
Address: 192.168.2.3
```

```
vagrant@vm1:~$ nslookup ns1.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.2.1
Address:        192.168.2.1#53

Name:   ns1.net1.as.uca.es
Address: 192.168.2.1
```

Resolución inversa:

Nota: Se indica que utilice el servidor para que muestre correctamente el nombre.

```
vagrant@vm1:~$ host 192.168.2.1 192.168.2.1
Using domain server:
Name: 192.168.2.1
Address: 192.168.2.1#53
Aliases:

1.2.168.192.in-addr.arpa domain name pointer ns1.net1.as.uca.es.
```

```
vagrant@vm1:~$ host 192.168.2.2 192.168.2.1
Using domain server:
Name: 192.168.2.1
Address: 192.168.2.1#53
Aliases:

2.2.168.192.in-addr.arpa domain name pointer vm1.net1.as.uca.es.
```

```
vagrant@vm1:~$ host 192.168.2.3 192.168.2.1
Using domain server:
Name: 192.168.2.1
Address: 192.168.2.1#53
Aliases:

3.2.168.192.in-addr.arpa domain name pointer vm2.net1.as.uca.es.
```

Mediante el comando dig:

```
vagrant@vm1:~$ dig vm2.net1.as.uca.es @192.168.2.1

; <<>> DiG 9.11.3-1ubuntu1.17-Ubuntu <<>> vm2.net1.as.uca.es @192.168.2.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37944
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e820490d40f04177c4c26f76625892927cc834e9d3657631 (good)
;; QUESTION SECTION:
;vm2.net1.as.uca.es.            IN      A

;; ANSWER SECTION:
vm2.net1.as.uca.es.     604800  IN      A       192.168.2.3

;; AUTHORITY SECTION:
net1.as.uca.es.         604800  IN      NS      ns1.net1.as.uca.es.

;; ADDITIONAL SECTION:
ns1.net1.as.uca.es.     604800  IN      A       192.168.2.1

;; Query time: 2 msec
;; SERVER: 192.168.2.1#53(192.168.2.1)
;; WHEN: Thu Apr 14 21:30:58 UTC 2022
;; MSG SIZE  rcvd: 125
```

Desde los hosts de la red 2 :

Nota: Sólo se adjuntarán capturas del primer host, puesto que el segundo es idéntico y por lo tanto, sería información redundante.

Resolución directa:

```
vagrant@vm3:~$ nslookup vm3.net2.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.3.1
Address:        192.168.3.1#53

Name:   vm3.net2.as.uca.es
Address: 192.168.3.2
```

```
vagrant@vm3:~$ nslookup vm4.net2.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.3.1
Address:        192.168.3.1#53

Name:   vm4.net2.as.uca.es
Address: 192.168.3.3
```

```
vagrant@vm3:~$ nslookup ns2.net2.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.3.1
Address:        192.168.3.1#53

Name:   ns2.net2.as.uca.es
Address: 192.168.3.1
```

Resolución inversa:

Nota: Se indica que utilice el servidor para que muestre correctamente el nombre.

```
vagrant@vm3:~$ host 192.168.3.1 192.168.3.1
Using domain server:
Name: 192.168.3.1
Address: 192.168.3.1#53
Aliases:

1.3.168.192.in-addr.arpa domain name pointer ns2.net2.as.uca.es.
```

```
vagrant@vm3:~$ host 192.168.3.2 192.168.3.1
Using domain server:
Name: 192.168.3.1
Address: 192.168.3.1#53
Aliases:

2.3.168.192.in-addr.arpa domain name pointer vm3.net2.as.uca.es.
```

```
vagrant@vm3:~$ host 192.168.3.3 192.168.3.1
Using domain server:
Name: 192.168.3.1
Address: 192.168.3.1#53
Aliases:

3.3.168.192.in-addr.arpa domain name pointer vm4.net2.as.uca.es.
```

Mediante el comando dig:

```
vagrant@vm3:~$ dig vm4.net2.as.uca.es @192.168.3.1

; <<>> DiG 9.11.3-1ubuntu1.17-Ubuntu <<>> vm4.net2.as.uca.es @192.168.3.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37044
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 688de975ecc5465cff2cecb0625893019c7f3e2bc469b2d8 (good)
;; QUESTION SECTION:
;vm4.net2.as.uca.es.            IN      A

;; ANSWER SECTION:
vm4.net2.as.uca.es.     604800  IN      A       192.168.3.3

;; AUTHORITY SECTION:
net2.as.uca.es.         604800  IN      NS      ns2.net2.as.uca.es.

;; ADDITIONAL SECTION:
ns2.net2.as.uca.es.     604800  IN      A       192.168.3.1

;; Query time: 2 msec
;; SERVER: 192.168.3.1#53(192.168.3.1)
;; WHEN: Thu Apr 14 21:32:49 UTC 2022
;; MSG SIZE  rcvd: 125
```

Y además, también funciona la resolución **DNS** hacia host de distinta red, por ejemplo, desde **vm3** puedo preguntar por **x.net1.as.uca.es**:

```
vagrant@vm3:~$ nslookup vm1.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.3.1
Address:        192.168.3.1#53

Name:   vm1.net1.as.uca.es
Address: 192.168.2.2
```

```
vagrant@vm1:~$ nslookup vm4.net2.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:         192.168.2.1
Address:        192.168.2.1#53

Name:   vm4.net2.as.uca.es
Address: 192.168.3.3
```

Esto es porque ambas zonas (**net1 y net2**) se encuentran conectadas al equipo router, por lo que, desde cualquier host (bajo la restricción **"trusted"** de la configuración) se puede consultar cualquier nombre de dominio.