ANÁLISIS DE SERVICIOS DE RED

Práctica SL3

Informe de prácticas

1. **Determine de forma experimental en qué difiere la información que recopila `tcpdump` cuando estamos trazando el tráfico de red y se produce un intento de conexión a un puerto de un servidor que está abierto y cuando se intenta conectar a uno que está cerrado.**

   tcpdump es una herramienta para línea de comando que sirve para analizar el tráfico de la red. Permite capturar y mostrar los paquetes que circulan por la red a la cual el sistema está conectado. Esta herramienta está instalada por defecto en la mayoría de sistemas UNIX. Para la captura de paquetes se apoya en la biblioteca *libpcap.*

   libpcap es una interfaz independiente del sistema que ofrece la utilidad de captura de paquetes a nivel de usuario. Proporciona un framework para monitoreo de red a bajo nivel. Se utiliza para la recopilación de estadísticas de red, control de seguridad, depuración de red, captura de paquetes que circulan por la red, etc.

   Por tanto, ***tcpdump*** es la orden por línea de comandos que permite el uso de esta librería. La forma más simple para poner en marcha la captura de paquetes hacia un host y puerto concretos de la red es la siguiente:

   sudo tcpdump host ‘hostname’ and port ‘portnumber’

   Donde ‘hostname’ es el nombre del sistema a vigilar. El uso de esta orden requiere de privilegios de administrador.

   Para tratar de ilustrar con un ejemplo la traza de qué es lo que sucede al tratar de acceder a un puerto abierto y a uno que está cerrado, nos apoyaremos en la orden ***nmap***.

   nmap es una herramienta para línea de comandos que sirve para efectuar rastreo de puertos. Se usa para evaluar la seguridad de sistemas informáticos, así como para descubrir servicios o servidores en una red informática, para ello envía unos paquetes definidos a otros equipos y analiza sus respuestas. También se puede utilizar para detección de sistemas conectados a una red, así como conocer servicios y sistemas operativos ejecutándose en la misma. Un ejemplo de uso del mismo podría ser:

   nmap -v -p ‘portnumbers’ ‘hostname’

   Donde ‘hostname’ es el nombre del sistema del cual se pretende obtener información y ‘portnumbers’ el número de los puertos de los cuales se pretende conocer su estado.

Con estas órdenes ya se puede llevar a cabo la prueba propuesta por el enunciado. Se ha decidido ejecutar en **mallet** la orden ***tcpdump*** que recopilará los paquetes, y será desde **bob** desde donde se tratará de conocer puertos abiertos en **alice** apoyándose en la utilidad ***nmap***. Los puertos elegidos para las pruebas son el **22** (ya que es el que se utiliza para conexiones ssh y sabemos que está abierto) y el **979** (que presuponemos que estará cerrado).

Para empezar a recopilar paquetes ejecutamos el siguiente comando en **mallet**:

sudo tcpdump port 979 or port 22

A continuación ejecutamos el escaneo de puertos desde **bob** hacia **alice**:

nmap –p 22, 979 alice

Los resultados obtenidos son los siguientes:

```
bob:~# nmap -p 22,979 alice

Starting Nmap 4.62 ( http://nmap.org ) at 2016-10-26 11:14 CEST
Interesting ports on alice (192.168.1.2):
PORT    STATE  SERVICE
22/tcp  open   ssh
979/tcp closed unknown
MAC Address: 08:00:27:ED:5B:F5 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.103 seconds
```

Como preveíamos el puerto **22** está abierto y el **979** cerrado, el siguiente paso es ver los resultados en **mallet** (en la práctica se ha escaneado los puertos con dos ejecuciones separadas para diferenciar más fácilmente lo que sucede en cada caso). Tampoco se ha utilizado el modo verbose en la prueba ya que se ha visto que en este caso no aportaba más información:

```
mallet@mallet:~$ sudo tcpdump port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:27:00.814687 IP bob.38522 > alice.ssh: Flags [S], seq 579697808, win 1024, options [mss 1460], length 0
11:27:00.814791 IP alice.ssh > bob.38522: Flags [S.], seq 956101434, ack 579697809, win 5840, options [mss 1460], length 0
11:27:00.814887 IP bob.38522 > alice.ssh: Flags [R], seq 579697809, win 0, length 0
```

Lo que sucede en el caso de un puerto abierto es que el host destino (alice) envía un paquete para indicarlo, que será para iniciar una conexión de comunicación mediante ssh, es decir, es un **SYN/ACK**. El último mensaje enviado por bob probablemente sea una cancelación de dicha conexión, por lo tanto, por la red fluyen 3 paquetes.

```
mallet@mallet:~$ sudo tcpdump port 979
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:26:23.898342 IP bob.44429 > alice.979: Flags [S], seq 2861228234, win 3072, options [mss 1460], length 0
11:26:23.898409 IP alice.979 > bob.44429: Flags [R.], seq 0, ack 2861228235, win 0, length 0
```

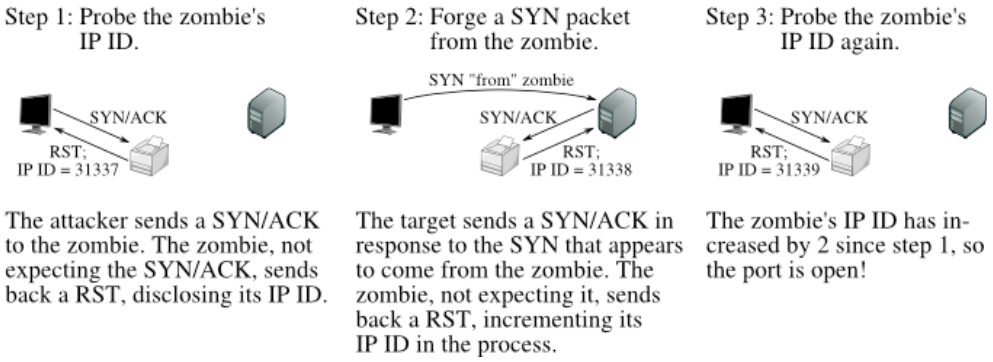
En el caso de un puerto cerrado esto no sucede, sino que únicamente se envía un paquete de respuesta, de tipo **RST**, para indicar que el puerto está cerrado. En este caso tan solo fluyen 2 paquetes.

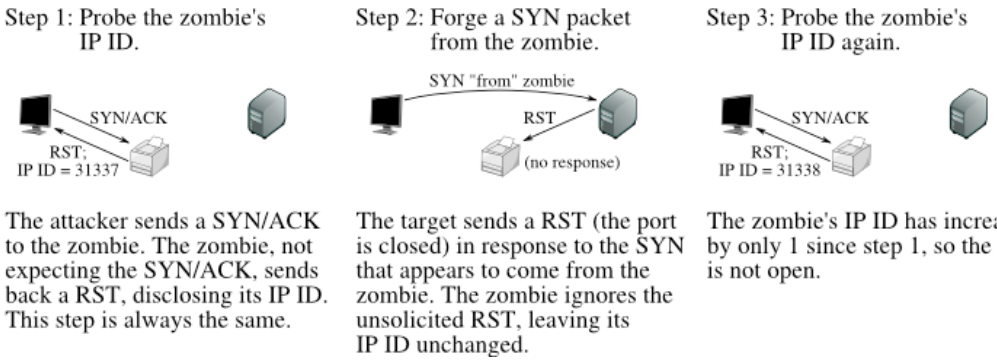
2. **Describa de forma clara y concisa en qué consiste un barrido de puertos TCP en modo 'sigiloso' (*stealth*) y en modo 'ocioso' (*idle*), incluyendo en su respuesta las referencias a las fuentes consultadas e ilustrando, a poder ser, con un ejemplo sobre la infraestructura virtualizada de que disponemos en la asignatura.**

Para el barrido de puertos TCP existen distintas técnicas para tratar de pasar desapercibido ante el host que se desea explorar. Los dos que se piden describir son el modo sigiloso y el modo ocioso:

Modo Sigiloso: El modo sigiloso consiste en un modo de escaneo que tan solo obtiene respuesta en los casos en los cuales el puerto se encuentra cerrado. Para ello se envía un paquete con las banderas FIN, URG y PSH. En el caso de que el puerto esté cerrado la respuesta será un SYN/RST. Para que este ataque sea exitoso el host objetivo deberá implementar la versión de TCP/IP indicada en el RFC-793. Para activar este modo en ***nmap*** hay que añadir a la orden '-sX'.

Modo Ocioso: Consiste en emular que la petición de acceso al puerto proviene de otro host 'zombie'. Para ello se obtiene el IP ID del zombie enviando un SYN/ACK. Después el host atacante utiliza este IP ID para enviar el SYN/ACK al host objetivo, pero camuflándolo como si el remitente fuera el host zombie. Seguidamente el host objetivo le enviará el paquete de respuesta y en el caso de que el puerto esté abierto el zombie incrementará en 1 su SYN/ACK. Para que el host atacante conozca el estado del puerto tan solo tiene que volver a realizar la primera tarea y en el caso de que el IP ID se haya incrementado en 1, el puerto estará cerrado, pero si ha sido en 2 significará que el puerto ha sido abierto. Para activar este modo en ***nmap*** hay que añadir a la orden '-sI'. Este concepto se ilustra más claramente en las siguientes imágenes:





Ahora se explicará el ejemplo que se ha llevado a cabo para practicar los conceptos descritos. Utilizaremos los mismos puertos que en el ejemplo anterior (22 y 979). En el caso del modo sigiloso realizaremos una conexión desde **mallet** hacia **alice** a través de ***nmap*** y utilizaremos ***tcpdump*** para monitorear lo ocurrido.

Primero realizaremos la simulación haciendo uso del modo sigiloso:

```
mallet@mallet:~$ sudo nmap -sX -p 22 alice

Starting Nmap 5.00 ( http://nmap.org ) at 2016-10-27 17:10 CEST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns o
r specify valid servers with --dns-servers
Interesting ports on alice (192.168.1.2):
PORT   STATE         SERVICE
22/tcp open|filtered ssh
MAC Address: 08:00:27:ED:5B:F5 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.29 seconds
```

```
mallet@mallet:~$ sudo tcpdump port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:09:18.350480 IP mallet.local.51872 > alice.ssh: Flags [FPU], seq 659400639, win 3072, urg 0, length 0
17:09:18.450883 IP mallet.local.51873 > alice.ssh: Flags [FPU], seq 659335102, win 3072, urg 0, length 0
```

Como vemos en el caso de que el puerto se encuentre abierto no se obtiene respuesta por parte del host objetivo. El motivo por el cual se envían dos paquetes es para asegurarse de que la causa no ha sido que el primero se haya perdido. También cabe destacar que ahora **nmap** indica en el estado abierto y además 'filtered'. Esto es debido a que como no ha obtenido una respuesta no puede examinar con exactitud cuál es el estado exacto.

```
mallet@mallet:~$ sudo nmap -sX -p 979 alice

Starting Nmap 5.00 ( http://nmap.org ) at 2016-10-27 17:11 CEST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns o
r specify valid servers with --dns-servers
Interesting ports on alice (192.168.1.2):
PORT    STATE  SERVICE
979/tcp closed unknown
MAC Address: 08:00:27:ED:5B:F5 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

```
mallet@mallet:~$ sudo tcpdump port 979
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:11:41.039309 IP mallet.local.37923 > alice.979: Flags [FPU], seq 2024936000, win 2048, urg 0, length 0
17:11:41.039681 IP alice.979 > mallet.local.37923: Flags [R.], seq 0, ack 2024936001, win 0, length 0
```

En este caso el puerto está cerrado, y como se puede observar a través de tcpdump el host objetivo envía un paquete de respuesta con la cabecera RST indicándolo.

Tras tratar de llevar a cabo la simulación en modo ocioso, no se ha podido llevar a cabo debido a que no se ha encontrado ningún puerto ni en alice ni en bob que devolviese el IP ID distinto de cero, por lo tanto no existe la vulnerabilidad necesaria para poder realizar el ataque. Aún así el comando que se usaría es:

sudo nmap –PN –p 'puerto' –sI 'host-zombie' 'host-objetivo'

'-PN' indica que no se quiere enviar mensajes ping, puerto es el puerto a explorar, 'host-zombie' el host desde el que se quiere simular el escaneo y 'host-objetivo' al que se quiere escanear.

REFERENCIAS:

- https://nmap.org/book/man-port-scanning-techniques.html
- https://nmap.org/book/idlescan.html
- https://www.linkedin.com/pulse/20140930164834-1571978-syn-stealth-xmas-null-idle-fin

3. **Describa paso a paso un ataque de saturación de buffer al puerto `12345` de `bob` desde `mallet`, desde el descubrimiento de la vulnerabilidad hasta la ejecución de una sesión de terminal remoto como usuario privilegiado en `bob`.**

Para realizar un ataque de saturación de buffer lo primero que hay que hacer es conocer el puerto que se desea utilizar para el ataque. Por lo tanto, utilizaremos ***nmap*** para conocer los puertos abiertos en **bob** para comprobar si existe algo inusual:

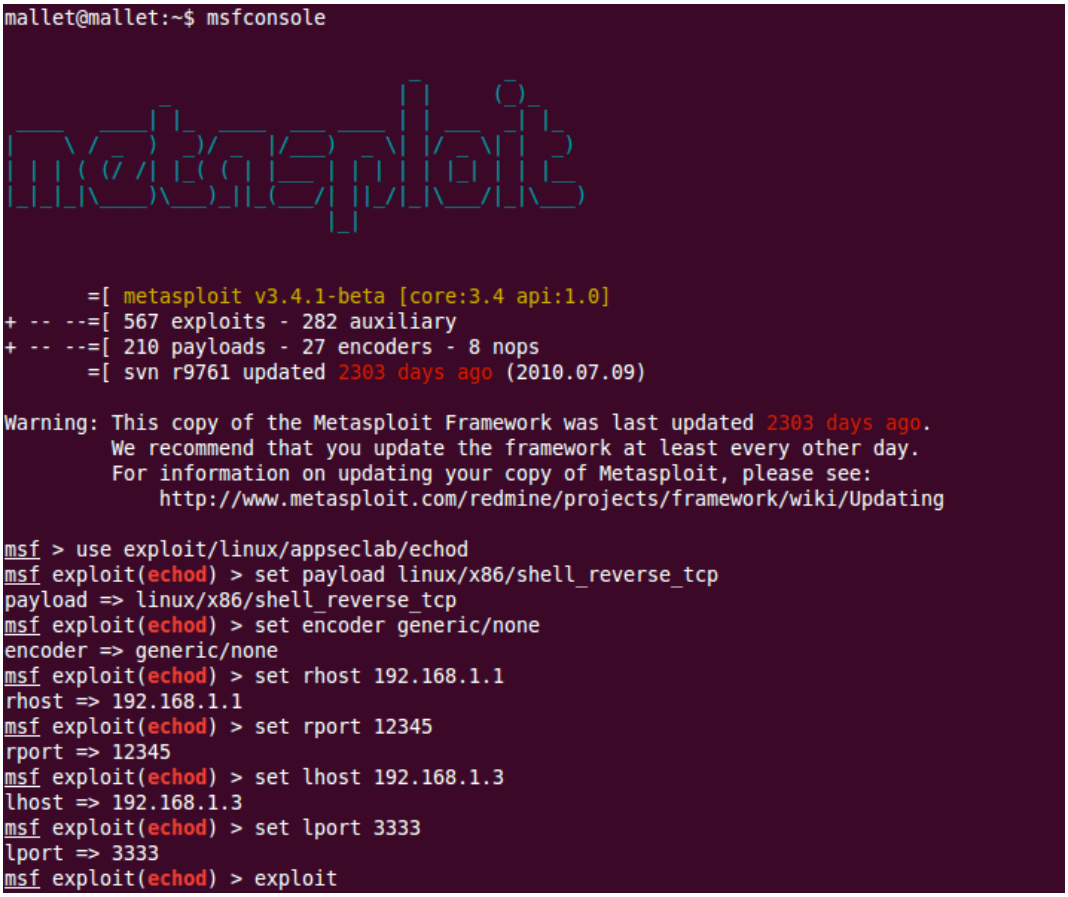
```
mallet@mallet:~/Exploits/Echo Daemon$ nmap bob

Starting Nmap 5.00 ( http://nmap.org ) at 2016-10-28 11:15 CEST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disab
led. Try using --system-dns or specify valid servers with --dns-servers
Interesting ports on bob (192.168.1.1):
Not shown: 995 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http
12345/tcp open  netbus

Nmap done: 1 IP address (1 host up) scanned in 0.09 seconds
```

Como se puede ver en la imagen adjunta, **bob** tiene abierto un puerto inusualmente extraño (12345) por lo que este podría ser usado para realizar el ataque. Para realizar dicha labor nos podemos apoyar en la orden ***msfconsole.***

msfconsole es, una de las interfaces más populares para el uso de Metasploit Framework (una herramienta para desarrollar y ejecutar exploits contra una máquina remota) a través de la línea de comandos. Proporciona una consola centralizada que permite un acceso eficaz a prácticamente todas las opciones disponibles en el framework.

Para este ataque lo que se pretende conseguir es un desbordamiento de buffer en el puerto encontrado para terminar consiguiendo acceder a un Shell con privilegios de administrador en el host **bob**, a través del cuál tengamos acceso a pleno a todas las configuraciones y ficheros.

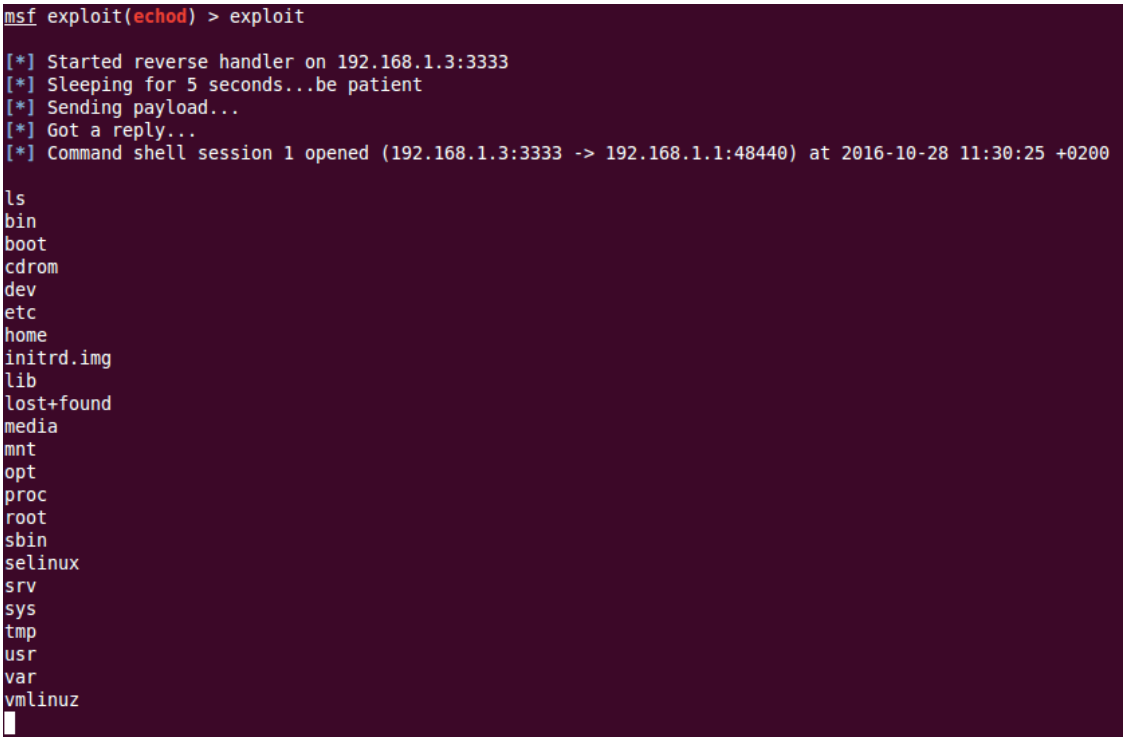
El proceso se ha ilustrado en la siguiente imagen:



Lo primero es seleccionar el tipo de vulnerabilidad (exploit) además de seleccionar lo que se espera conseguir (payload). Lo siguiente es seleccionar la IP y puerto del host objetivo, así como del host desde donde se quiere explotar dicha vulnerabilidad.

Una vez fijados todos estos parámetros ya se puede llevar a cabo el ataque ejecutando la sentencia '*exploit*'. A partir de este momento lo que aparece es un

nuevo Shell que se corresponde con el host objetivo. Esto se ilustra en la siguiente imagen. En este caso se está ejecutando la orden '*ls*', que muestra todos los ficheros y directorios del host objetivo (**bob** en este caso):



4. **Describa tres mecanismos básicos de protección contra intrusos que estén basados en el control de acceso.**

Los sistemas implementan tres mecanismos básicos de protección contra intrusos a través de la red, basados en control de acceso. Con dichas estrategias se pretende denegar el acceso a usuarios ilegítimos o con intenciones poco claras. Los tres mecanismos son los siguientes:

Firewall:
Es un sistema de seguridad de red que monitorea y controla el tráfico de red entrante y saliente basándose en reglas de seguridad previamente preestablecidas. Normalmente se usa para establecer una barrera entre una red interna confiable y segura con otra red externa, como Internet, que se supone que no es segura.

Los cortafuegos se pueden clasificar en dos tipos:

Los *firewalls de red* filtran el tráfico entre dos o más redes. Son aplicaciones de software que se ejecutan en hardware de propósito general o dispositivos de computadora de firewall basados en hardware.

Los *firewalls basados en host* proporcionan una capa de software en un host que controla el tráfico de red dentro y fuera de esa única máquina.

Los firewalls se compilan normalmente en el kernel o se agregan como un módulo del mismo. El cortafuegos más destacado en máquinas Unix es netfilter/iptables.

Para visualizar el estado del firewall de una máquina podemos hacer uso del siguiente comando (en este caso no existía ninguna restricción en el firewall):

```
alice@alice:~$ sudo iptables -L -n -v
Chain INPUT (policy ACCEPT 4 packets, 200 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 4 packets, 200 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Para demostrar su uso limitaremos el acceso del puerto 12345, que como vimos en el ejercicio anterior era vulnerable ante ataques de desbordamiento de buffer y solo permitiremos su acceso desde bob:

```
alice@alice:~$ sudo iptables -A INPUT -p tcp --dport 12345 -s 192.168.1.1 -j ACCEPT
alice@alice:~$ sudo iptables -A INPUT -p tcp --dport 12345 -j DROP
```

Como vemos la regla se ha añadido correctamente al firewall:

```
alice@alice:~$ sudo iptables -L -n -v
Chain INPUT (policy ACCEPT 242 packets, 13252 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       192.168.1.1          0.0.0.0/0           tcp dpt:12345
    0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:12345

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 242 packets, 13252 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

TCP Wrapper:
Es un sistema de ACL (listas de control de acceso) de red basado en host, utilizado para filtrar el acceso a los puertos TCP del mismo. La característica diferenciadora respecto de la alternativa anterior es que en este caso se ejecuta en el espacio de usuarios y no en el kernel como el firewall.

Las restricciones de denegación y permiso de acceso se almacenan en los ficheros */etc/hosts.deny* y */etc/hosts.allow* respectivamente. Estos ficheros se escriben siguiendo una estructura estándar, que se describe en la entrada del manual denominada ***hosts_access***. La estructura básica es la siguiente:

daemon : client [:option1:option2:..]

Un ejemplo de uso de esta estrategia se llevará a cabo en el ejercicio siguiente.

Configuración:
También existen mecanismos de control de acceso basados en la configuración del propio servicio al que se refieren. Existen muchos ejemplos de ello, como el servidor ***ssh***, que permite añadir restricciones de permiso o denegación a usuarios y grupos concretos. El servicio para el servidor ***Apache*** también proporciona su propio mecanismo de control de acceso mediante ficheros de configuración.

Para ilustrar su uso se ha denegado el acceso tanto del usuario como del grupo mediante **ssh** a la máquina Alice añadiendo las siguientes líneas al fichero */etc/ssh/sshd_config:*

```
DenyUsers root
DenyGroups root
                                                          90,15         Bot
```

5. **Indique cómo impediría acceso al servicio FTP de bob desde cualquier host que no fuese Alice y muestre que funciona (captura de pantalla)**

Para restringir el acceso al servicio FTP de bob a todos los host que pretendan acceder a él excepto a *Alice* primero comprobaremos el estado actual del servicio tratando de acceder tanto desde *Alice* como desde *Mallet*:

```
alice@alice:~$ ftp bob
Connected to bob.
220 ProFTPD 1.3.0 Server (ProFTPD Default Installation) [192.168.1.1]
Name (bob:alice): bob
331 Password required for bob.
Password:
230 User bob logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
-rw-r--r--   1 bob      bob          1750 Sep 22  2010 echod.c
-rw-r--r--   1 bob      bob          3351 Sep 22  2010 kernel_root_sock_sendpage
.c
-rw-r--r--   1 bob      bob         10416 Sep 22  2010 kernel_root_udp_sendmsg.c
-rw-r--r--   1 bob      bob          5834 Sep 22  2010 kernel_root_vmsplice.c
-rwxr-xr-x   1 bob      bob          1613 Sep 22  2010 proftpd_local_root.py
226 Transfer complete.
ftp>
```

```
mallet@mallet:~$ ftp bob
Connected to bob.
220 ProFTPD 1.3.0 Server (ProFTPD Default Installation) [192.168.1.1]
Name (bob:mallet): bob
331 Password required for bob.
Password:
230 User bob logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
-rw-r--r--   1 bob      bob          1750 Sep 22  2010 echod.c
-rw-r--r--   1 bob      bob          3351 Sep 22  2010 kernel_root_sock_sendpage
.c
-rw-r--r--   1 bob      bob         10416 Sep 22  2010 kernel_root_udp_sendmsg.c
-rw-r--r--   1 bob      bob          5834 Sep 22  2010 kernel_root_vmsplice.c
-rwxr-xr-x   1 bob      bob          1613 Sep 22  2010 proftpd_local_root.py
226 Transfer complete.
ftp>
```

Como vemos, se puede acceder desde los dos hosts sin ningún problema. Puesto que la estrategia de uso **TCP Wrappers** requiere de configuración avanzada cuando se usa con el servidor **FTP** de *Bob* (***proftpd***) se utilizará el firewall (***iptables***) para denegar el acceso. Esto se ha conocido mediante la ejecución de la siguiente orden y una búsqueda por la red sobre cómo configurar dicho servidor ftp:

```
bob:~# ps -ef | grep -i ftp
nobody    1107     1  0 17:42 ?        00:00:00 proftpd: (accepting connections)
bob       1309  1107  0 17:42 ?        00:00:00 proftpd: bob - alice: IDLE
bob       1310  1107  0 17:42 ?        00:00:00 proftpd: bob - mallet: IDLE
root      1312  1303  0 17:46 tty1     00:00:00 grep -i ftp
```

Por lo tanto, las órdenes que necesitamos ejecutar para restringir el acceso del servidor **FTP** para que solo *alice* lo pueda usar son las siguientes:

```
bob:~# iptables -A INPUT -p tcp --dport ftp -s alice -j ACCEPT
bob:~# iptables -A INPUT -p tcp --dport ftp -j DROP
```

Por lo que la ACL de *bob* queda de la siguiente manera:

```
bob:~# iptables -L -n -v
Chain INPUT (policy ACCEPT 6 packets, 314 bytes)
 pkts bytes target     prot opt in     out     source               destination

   13   741 ACCEPT     tcp  --  *      *       192.168.1.2          0.0.0.0/0
        tcp dpt:21
   12   696 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
        tcp dpt:21

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 17 packets, 1525 bytes)
 pkts bytes target     prot opt in     out     source               destination

bob:~# _
```

Por último, tratamos de realizar una conexión tanto desde *alice* como desde mallet permitiéndo el acceso correcto y sin problemas desde el primero, pero, por el contrario, denegándolo desde el segundo:

```
alice@alice:~$ ftp bob
Connected to bob.
220 ProFTPD 1.3.0 Server (ProFTPD Default Installation) [192.168.1.1]
Name (bob:alice): bob
331 Password required for bob.
Password:
230 User bob logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
-rw-r--r--   1 bob      bob          1750 Sep 22  2010 echod.c
-rw-r--r--   1 bob      bob          3351 Sep 22  2010 kernel_root_sock_sendpage.c
-rw-r--r--   1 bob      bob         10416 Sep 22  2010 kernel_root_udp_sendmsg.c
-rw-r--r--   1 bob      bob          5834 Sep 22  2010 kernel_root_vmsplice.c
-rwxr-xr-x   1 bob      bob          1613 Sep 22  2010 proftpd_local_root.py
226 Transfer complete.
ftp>
```

```
mallet@mallet:~$ ftp bob
ftp: connect: Connection timed out
ftp>
```