UNPSJB

LICENCIATURA EN SISTEMAS OPGCPI

ADMINISTRACIÓN DE REDES Y SEGURIDAD

Trabajo Práctico 2

Footprinting - Fingerprinting - Escaneo - Enumeración - Firewall

---

Cátedra
Lic. Bruno Damián Zappellini

Integrantes:
Luciano Serruya Aloisi

13 de octubre de 2018

Índice

1. *Footprinting*

El término *Footprinting* se refiere al proceso de recolectar la mayor cantidad de información posible sobre un sistema objetivo con el fin de encontrar formas de penetrarlo [6]. Este etapa previa a realizar un ataque, conocida como *fase de reconocimiento*, el atacante intenta encontrar información como [7]:

- Rango de Red y sub-red (*Network Range* y *subnet mask*)
- Acertar máquinas o computadoras activas
- Puertos abiertos y las aplicaciones que están corriendo en ellos
- Detectar versiones de sistemas operativos
- Nombres de Dominios (*Domain Names*)
- Bloques de Red (*Network Blocks*)
- Direcciones IP específicas
- País y ciudad donde se encuentran los servidores
- Información de contacto (números telefónicos, emails)
- *DNS records*

Mucha de la información antes mencionada, como Domain Names, algunas direcciones IP, país, ciudad, e información de contacto puede ser conseguida consultando a las bases de datos de *whois*. Esto se realiza justamente con el comando `whois` y el nombre del *dominio* al cual se quiere consultar. Por ejemplo, si se desea conocer información sobre el dominio *facebook.com*, se debe realizar la siguiente invocación a `whois`:

```
1    whois facebook.com
```

Además del comando *whois*, que recupera información detallada sobre el dominio consultado (quién es su dueño, fecha de registro, fecha de expiración, entre otros), otras herramientas para hacer consultas a DNS son los comandos `nslookup` y `dig`. Para hacer *enumeración de DNS* (obtener todos los subdominios registrados bajo un dominio) existen herramientas como `fierce`, `dnsrecon`, o `dnsenum` [2].

Elija dos organizaciones cualesquiera y utilizando WHOIS y DIG, averigüe toda la información que pueda: servidores de correo, servidores DNS, Servidores WEB, etc

Dentro del directorio “assets” se incluye un *script* nombrado “footprinting.sh”, el cual recibe un nombre de dominio y realiza varias consultas con los comando `dig` y `whois`. A continuación se incluye un ejemplo de ejecución con el dominio *github.com* y las partes más importantes de su salida

```
1   bash footprinting.sh github.com
2
3   >>>
4
5    *** dig -t NS +short github.com ***
6    ns3.p16.dynect.net.
7    ns1.p16.dynect.net.
8    ns4.p16.dynect.net.
9    ns-520.awsdns-01.net.
10   ns-1283.awsdns-32.org.
11   ns2.p16.dynect.net.
12   ns-1707.awsdns-21.co.uk.
13   ns-421.awsdns-52.com.
14
15   *** dig -t MX +short github.com ***
16   1 ASPMX.L.GOOGLE.com.
17   5 ALT1.ASPMX.L.GOOGLE.com.
18   5 ALT2.ASPMX.L.GOOGLE.com.
19   10 ALT3.ASPMX.L.GOOGLE.com.
20   10 ALT4.ASPMX.L.GOOGLE.com.
21
22   *** dig -t SOA +short github.com ***
23   ns1.p16.dynect.net. hostmaster.github.com. 1538412644
        3600 600 604800 60
24
25   *** whois github.com ***
26      Domain Name: GITHUB.COM
27      Registry Domain ID: 1264983250_DOMAIN_COM-VRSN
28      Registrar WHOIS Server: whois.markmonitor.com
```

```
29          Registrar URL: http://www.markmonitor.com
30          Updated Date: 2017-06-26T16:02:39Z
31          Creation Date: 2007-10-09T18:20:50Z
32          Registry Expiry Date: 2020-10-09T18:20:50Z
33          .
34          .
35          .
```

Visite el sitio http://www.netcraft.net/ y pruebe la funcionalidad del mismo contra el dominio www.unp.edu.ar

Algunos de los datos que indica sitio *www.netcraft.com* sobre el dominio de la UNP son los siguientes:

- Título del sitio: *Universidad Nacional de la Patagonia San Juan Bosco*
- Visto por primera vez en *Junio de 1998*
- Lenguaje primario *español*
- Puntaje de 7 sobre 10 en *Netcraft Risk Rating* [1]
- Dominio *unp.edu.ar*
- Dirección IPv4 *170.210.88.21*
- *Nameserver chenque.unp.edu.ar*
- Administrador de DNS *hostmaster@unp.edu.ar*

Visite el sitio http://www.archive.org/web/web.php y pruebe la funcionalidad del mismo contra el sitio web de la UNP: www.unp.edu.ar. ¿Qué ventajas presenta esta herramienta respecto de otras herramientas de footprinting?

A diferencia de herramientas como `dig` y `whois`, el sitio *www.archive.org* se dedica a visitar sitios web y tomarles un *snapshot* de su estado actual. Al hacerle una consulta

[1] Aunque algunos sitios tengan contenido no malicioso, *Netcraft Extension* puede asignar un valor alto de riesgo porque está siendo servido bajo un dominio recientemente agregado a la base de datos de *Netcraft*, porque el sitio nunca fue visto en la *Netcraft Web Server Survey*, o porque la red que sirve el sitio ha servido sitios fraudulentos en el pasado. Distintos factores son tomados en cuenta [5]

sobre algún sitio en particular, muestra los distintos cambios por los cuales ha pasado, pudiendo ver versiones anteriores. También brinda herramientas para visualizar la cantidad de archivos tipo MIME con los cuales ha contado el sitio (ya sean imágenes, hojas de estilo, archivos con código Javascript, y demás). Por último, recolecta y muestra las distintas *URLs* que publica el sitio, con los recursos a los cuales se pueden acceder a través de la *URL*.

2. *Fingerprinting* de servidores

- El sitio *www.google.com* utiliza como servidor web *Google Web Server* (*GWS*), pero no se puede saber por las cabeceras HTTP de la respuesta qué versión de servidor usa
- El sitio *www.ing.unp.edu.ar* indica que está usando la versión 1.10.3 del servidor web *NginX*
- El sitio *www.microsoft.com* utiliza como servidor web *Apache*, pero no se puede saber por las cabeceras HTTP de la respuesta qué versión de servidor usa
- El sitio *serconex.juschubut.gov.ar* utiliza un servidor web *Microsoft-IIFS*, versión 10.0

3. Escaneo

El *escaneo* ó *scanning* es una actividad que consiste en detectar distintos dispositivos conectados a la red, pudiendo saber qué sistema operativo están corriendo, qué puertos tienen abiertos, o qué servidores están atendiendo peticiones.

- Escaneo de hosts: se puede realizar con `nmap` o con un pequeño script en *bash* que envíe un paquete utilizado `ping` a cada IP posible de la red (sabiendo la dirección de la red y su máscara, se puede calcular cuántas IPs habrán). De esta forma se puede averiguar cuántos dispositivos hay conectados a la red (aunque `nmap` brinda más información)
- Escaneo de puertos: también es posible hacerlo con `nmap`, indicándole una IP en particular (o combinándolo con *bash*, para que itere sobre varias dirección IP). De esta forma se puede saber qué puertos tiene abierto un *host* y qué servicios ofrece (también se podría intentar explotar alguna posible vulnerabilidad)

- Escaneo de redes WiFi: las placas modernas de red lo hacen automáticamente. Detectan qué redes WiFi hay en su rango de alcance, indicando su *SSID* (si es que es público), si requieren contraseña, con qué algoritmo de encriptación trabajan para manejar las contraseñas, entre otros
- Escaneo de dispositivos bluetooth: se podría hacer con un dispositivo Android. De esta forma se puede detectar qué otros dispositivos están a la escucha de conexiones bluetooth, y se podría de estar forma intentar explotar alguna vulnerabilidad de la versión de bluetooth que estén corriendo.

Indique qué tipo de escaneo (hosts, puertos, vulnerabilidades, WiFi) es posible realizar

- *Sólo manipulando el protocolo ARP: hosts* (misma red)
- *Sólo manipulando el protocolo ICMP: hosts* (no es necesario estar en la misma red)
- *Sólo manipulando el protocolo TCP:* puertos (no es necesario estar en la misma red).
- *Sólo manipulando el protocolo UDP:* puertos (no es necesario estar en la misma red).
- *Interpretando en forma pasiva tráfico de red (LAN o algún tipo de radiofrecuencias):* WiFi (misma red)

4. Escaneo de puertos

Para verificar los puertos abiertos en la máquina virtual de Kali, primero se ejecutó el comando `netstat` con dos combinaciones de parámetros distintas

```
root@kali:~# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign
   Address         State
```

Ningún puerto abierto

```
root@kali:~# netstat -nltp4
Active Internet connections (only servers)
```

```
3 Proto Recv-Q Send-Q Local Address           Foreign
    Address         State       PID/Program name
```

Ningún puerto abierto

Luego, la salida que se obtiene de ejecutar `nmap` es consistente con lo indicado por `netstat`

```
1 root@kali:~# nmap 127.0.0.1
2
3 Starting Nmap 7.40 ( https://nmap.org ) at 2018-10-05
    06:58 EDT
4 Nmap scan report for localhost (127.0.0.1)
5 Host is up (0.0000040s latency).
6 All 1000 scanned ports on localhost (127.0.0.1) are
    closed
7
8 Nmap done: 1 IP address (1 host up) scanned in 0.15
    seconds
```

```
1 root@kali:~# nmap -p- 127.0.0.1
2
3 Starting Nmap 7.40 ( https://nmap.org ) at 2018-10-05
    06:58 EDT
4 Nmap scan report for localhost (127.0.0.1)
5 Host is up (0.0000020s latency).
6 All 65535 scanned ports on localhost (127.0.0.1) are
    closed
7
8 Nmap done: 1 IP address (1 host up) scanned in 0.45
    seconds
```

Escaneo de los 65535 puertos disponibles

Ahora bien, se abrimos un puerto TCP con el comando `ncat`, la salida tanto de `netstat` como de `nmap` cambia acordemente

```
1 ncat -l 8080
```

Ponemos a escuchar al puerto 8080 por conexiones TCP

```
root@kali:~# netstat -nltp4
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign
  Address         State       PID/Program name
tcp        0      0 0.0.0.0:8080            0.0.0.0:*
                  LISTEN      2354/ncat
```

`netstat` indica un puerto abierto

```
root@kali:~# nmap -p- 127.0.0.1

Starting Nmap 7.40 ( https://nmap.org ) at 2018-10-05
  07:14 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000020s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
8080/tcp open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 0.58
  seconds
```

`nmap` indica 65534 puertos cerrados, y uno abierto

4.1. Pruebas con `hping3`

Escaneo del puerto TCP/80 de la máquina local (localhost)

- Comando a ejecutar: `hping3 -c 3 -p 80 -S localhost`
  - En la salida de `tcpdump` se puede ver que se está mandando un paquete TCP con la bandera `SYN` al puerto 80 (*Half-open SYN flag scanning*)
  - La respuesta a dicho envío es un paquete con las banderas `RST` y `ACK`; a partir de esa respuesta se puede deducir que **el puerto estaba cerrado**

Escaneo del puerto TCP/113 de la máquina local (localhost)

- Comando a ejecutar: `hping3 -c 3 -p 113 -S localhost`
    - En la salida de `tcpdump` se puede ver que se está mandando un paquete TCP con la bandera `SYN` al puerto 113 (*Half-open SYN flag scanning*)
    - La respuesta a dicho envío es un paquete con las banderas `RST` y `ACK`; a partir de esa respuesta se puede deducir que **el puerto estaba cerrado**

Escaneo del puerto UDP/631 de la máquina local (localhost)

- Comando a ejecutar: `hping3 -c 3 -p 631 -2 localhost`
    - En la salida de `tcpdump` se puede ver que se está mandando un paquete UDP al puerto 631 (*UDP ICMP port unreachable scanning*)
    - La respuesta a dicho envío es un mensaje ICMP indicando que **el puerto UDP 631 está inalcanzable**

Escaneo del puerto UDP/53 de la máquina local (localhost)

- Comando a ejecutar: `hping3 -c 3 -p 53 -2 localhost`
    - En la salida de `tcpdump` se puede ver que se está mandando un paquete UDP al puerto 53 (DNS) (*UDP ICMP port unreachable scanning*)
    - La respuesta a dicho envío es un mensaje ICMP indicando que **el puerto UDP 53 está inalcanzable**

4.2. *IDLE SCAN*

¿Qué características debe reunir un host que se pueda utilizar como zombie?
Para que un *host* sea utilizado como *zombie*, su tráfico en la red debe ser mínimo, de modo que el ID de los paquetes IP (*IPID*) sólo incremente con los paquetes enviados para el *IDLE SCAN* (o que el incremento sea predecible).

4.3. Detección de escaneo de puertos

Una forma de detectar y evitar escaneo de puertos, sería bloquear las direcciones IPs de aquellos *hosts* que estén realizando alguna actividad sospechosa como por ejemplo enviar un paquete TCP con las banderas `SYC+ACK` cuando no se estaban esperando.

Con respecto a la detección de un *idle scan*, se debería evitar el uso de valores de ID de los paquetes IP enviados predecibles.

Cabe aclarar que existen implementaciones tanto de software como de hardware (IDS) para detectar escaneo de puertos, de hosts, de vulnerabilidades, entre otros.

4.4. *Fingerprinting* de sistemas operativos

En [1] se da la siguiente definición de *fingerprinting de sistemas operativos*:

> (*OS Fingerprinting*) es el proceso de recopilación de información que permite identificar el sistema operativo en el ordenador que se tiene por objetivo. El **OS Fingerprinting activo** se basa en el hecho de que cada sistema operativo responde de forma diferente a una gran variedad de paquetes malformados. De esta manera, utilizando herramientas que permitan comparar las respuestas con una base de datos con referencias conocidas, es posible identificar cuál es el sistema operativo.
>
> [...] El **OS Fingerprinting pasivo** no se realiza directamente sobre el sistema operativo objetivo. Este método consiste en el análisis de los paquetes que envía el propio sistema objetivo a través de técnicas de sniffing. De esta forma, es posible comparar esos paquetes con una base de datos donde se tenga referencias de los distintos paquetes de los diferentes sistemas operativos y, por lo tanto, es posible identificarlos.

La finalidad de llevar a cabo un proceso de *fingerprinting de sistemas operativos* consiste en **determinar qué sistema operativo está corriendo** cierto *host* dentro de una red.

El comando `nmap` permite hacer *fingerprinting de sistemas operativos*. Para ello, se lo debe ejecutar con la bandera `-o`

```
1 $ sudo nmap -O <IP>
```

Se debe ejecutar el comando con privilegios de superusuario

```
1 $ sudo nmap -O 192.168.0.16
2
3 Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-05
    11:44 -03
4 Nmap scan report for 192.168.0.16
5 Host is up (0.0054s latency).
```

```
6  Not shown: 999 closed ports
7  PORT     STATE SERVICE
8  8080/tcp open  http-proxy
9  MAC Address: 30:3A:64:2C:0E:25 (Intel Corporate)
10 Device type: general purpose
11 Running: Linux 3.X|4.X
12 OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:
     linux_kernel:4
13 OS details: Linux 3.2 - 4.9
14 Network Distance: 1 hop
15
16 OS detection performed. Please report any incorrect
     results at https://nmap.org/submit/ .
17 Nmap done: 1 IP address (1 host up) scanned in 4.61
     seconds
18 [luciano@arch-bangho:~/Documents] bash $ sudo nmap -o
     192.168.0.16
19 Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-05
     11:45 -03
20 WARNING: No targets were specified, so 0 hosts scanned.
21 Nmap done: 0 IP addresses (0 hosts up) scanned in 0.03
     seconds
```

OS Fingerprinting a un Ubuntu 16.04

```
1  $ sudo nmap -O 192.168.0.48
2
3  Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-05
     11:47 -03
4  Nmap scan report for pcdebian1 (192.168.0.48)
5  Host is up (0.0045s latency).
6  Not shown: 999 closed ports
7  PORT   STATE SERVICE
8  22/tcp open  ssh
9  MAC Address: 84:16:F9:11:DA:02 (Tp-link Technologies)
10 Device type: general purpose
11 Running: Linux 3.X|4.X
```

```
12 OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:
     linux_kernel:4
13 OS details: Linux 3.2 - 4.9
14 Network Distance: 1 hop
15
16 OS detection performed. Please report any incorrect
     results at https://nmap.org/submit/ .
17 Nmap done: 1 IP address (1 host up) scanned in 2.43
     seconds
```

OS Fingerprinting a un Debian 9

4.5. *Fingerprinting* de servicios

¿Cuál es la finalidad de realizar fingerprinting de servicios? ¿banner grabbing es la forma más sencilla de realizarlo?

La finalidad del *fingerprinting de servicios* es **qué servicios y qué versiones de los servicios tiene corriendo un *host* en un puerto bien conocido**. Una de las formas más sencillas de realizarlo es a través del *banner grabbing*, que consiste en determinar el servicio y su versión en base a la respuesta inicial que dio el servidor al mensaje enviado al puerto bien conocido.

Un ejemplo de esto sería enviarle un paquete al puerto 22 de un servidor, y verificar qué servidor de SSH está atendiendo.

5. Enumeración

¿Qué es enumeración?

La enumeración tiene lugar después de la etapa de *escaneo* (*footprinting*, *fingerprinting*, *Google hacking*) y es el proceso de recopilación y compilación de nombres de usuario, nombres de las máquinas, recursos de red, recursos compartidos y servicios que se pueden conseguir de un objetivo [3].

5.1. Enumeración de distintos protocolos

- Redes *WiFi*

    - `nmcli dev wifi`
    - `sudo iw dev <INTERFAZ_INALÁMBRICA> scan`
- Dispositivos *bluetooth* (la computadora que realiza el escaneo debe contar con un dispositivo bluetooth y estar encendido)
    - `hcitool scan`
- Recursos presentes de un red Windows
    - `nbtscan <DIRECCION_RED>/<MÁSCARA>`
- Información de DNS de algún dominio en particular
    - `dig any +nocmd +multiline +noall +answer <DOMINIO>`

5.2. Enumeración de DNS con Kali Linux

Dentro del directorio "assets" se encuentra el archivo "dnsenum-unpeduar.txt" a modo de evidencia del experimento de enumeración de DNS con `dnsenum`.

6. Ataques de fuerza Bruta

Para realizar los siguientes experimentos, se configurún un servidor SSH en la máquina virtual de Kali, permitiendo ingresar como superusuario.

6.1. Atacando SSH con `msfconsole`

Dentro de la consola de *metasploit*, se ingresaron los siguientes comando para realizar un ataque de fuerza bruta al servidor SSH:

- `use scanner/ssh/ssh_login`
- `set RHOSTS 127.0.0.1`
- `set PASS_FILE badpasswords` (se incluye el archivo "badpasswords" dentro del directorio "assets")

- `set STOP_ON_SUCCESS true`
- `set USER root`

Figura 1: Ataque de fuerza bruta a un servidor SSH con `msfconsole`

6.2. Atacando SSH con `hydra-gtk`

Para realizar el mismo experimento pero con la herramienta `hydra-gtk` (versión con interfaz gráfica de la herramienta `hydra`), se hicieron los siguientes pasos:

- *Target*
  - `Single Target` > 192.168.0.4
  - `Port` > 22
  - `Protocol` > SSH
- *Passwords*
  - `Username` > root
  - `Password List` > badpasswords
- *Start*
  - `Start`

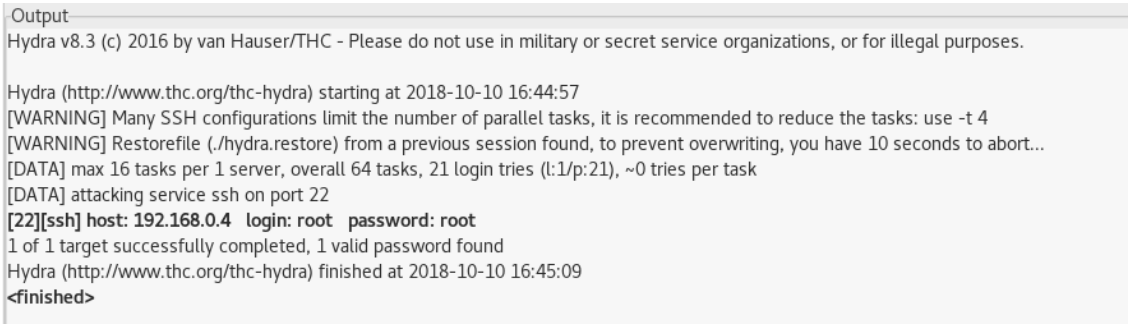

Figura 2: Ataque de fuerza bruta a un servidor SSH con `hydra-gtk`

7. *Firewall*

7.1. Ejemplo de implemetación de un *firewall*

¿Qué tipo de política de firewall se implementó?

El enunciado indica que el *firewall* de la red es sin estados y que **solamente debería permitir** dos ocasiones particulares, por lo tanto es una política **restrictiva** (bloquea todo aquello que no esté expresamente indicado).

¿Son suficientes estas reglas? En caso que no las considere suficientes para resolver el objetivo planteado, indique qué reglas agregaría y en qué orden las pondría

Con la primer regla se logra cumplir el objetivo de permitir a usuario de Internet (usuarios externos a la red) acceder al sitio web publicado (proceso corriendo en el puerto 80). Sin embargo, el segundo objetivo no está resuelto todavía; para ello, haría falta agregar la siguiente regla **antes de la definición de la política** (antes de la última regla):

Orden	Protocolo	IP Origen	Puerto Origen	IP Destino	Puerto Destino	Acción
2	TCP	ALL	ALL	200.10.11.2	443	Aceptar

Si ahora se quiere que la "Estación Y: 200.10.11.100" pueda hacer ping a www.google.com *para ver los tiempos de respuesta ¿cómo modificaría las reglas del firewall?*

Para lograr este nuevo objetivo, una solución sencilla sería añadir la siguiente regla **antes de la última**

Orden	Protocolo	IP Origen	Puerto Origen	IP Destino	Puerto Destino	Acción
3	TCP	200.10.11.100	ALL	64.233.190.105	443	Aceptar

7.2. *IPTables*

7.2.1. Topología *LAN*

Para automatizar la obtención de información sobre interfaces de red y estado de los puertos de cada *host* emulado con CORE, se utilizó el *script* "get-info.sh" (que se puede encontrar en el directorio "assets/iptables"). Las salidas de las ejecuciones en cada nodo se pueden encontrar en el directorio "assets/iptables/output-lan".

Al ejecutar el comando `iptables -nL -v` en el nodo *n1* (servidor), se obtiene la siguiente salida:

```
root@n1:/tmp/pycore.35517/n1.conf# iptables -nL -v
Chain INPUT (policy ACCEPT 0 packets, 0 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 0 packets, 0 bytes)
 pkts bytes target     prot opt in _   out     source               destination
```

Figura 3: Ningua regla configurada todavía

Casos de configuración

CASO 1: *Configure el firewall del Servidor Web para aceptar solamente conexiones al puerto 80 utilizando una política restrictiva*

- Configuración del servidor:
  - `iptables -P INPUT DROP`
  - `iptables -A INPUT -p tcp -dport 80 -j ACCEPT`
- Salida de `iptables -nL -v`

```
root@n1:/tmp/pycore.35517/n1.conf# iptables -nL -v
Chain INPUT (policy DROP 11 packets, 660 bytes)
 pkts bytes target     prot opt in     out     source               destination

   38  2147 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
        tcp dpt:80

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


Chain OUTPUT (policy ACCEPT 37 packets, 1960 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

CASO 2: *Configure el firewall del Servidor Web para aceptar solamente conexiones al puerto 80 utilizando una política permisiva*

- Configuración del servidor:
  - `iptables -P INPUT ACCEPT`
  - `iptables -A INPUT -p tcp -dport 0:79 -j DROP`
  - `iptables -A INPUT -p tcp -dport 81:65535 -j DROP`
- Salida de `iptables -nL -v`

```
root@n1:/tmp/pycore.35517/n1.conf# iptables -nL -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpts:0:79
    0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpts:81:65535
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80 state RELATED,ESTABLISHED

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

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

CASO 3: *Configure el firewall del Servidor Web para redireccionar toda petición al puerto TCP 8080 al puerto TCP 80 del mismo equipo*

- Configuración del servidor:
  - `iptables -t nat -A PREROUTING -p tcp -dport 8080 -j REDIRECT -to-port 80`

- Salida de `iptables -t nat -nL -v`

```
root@n1:/tmp/pycore.35517/n1.conf# iptables -t nat -nL -v
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 REDIRECT   tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:8080 redir ports 80

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

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

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@n1:/tmp/pycore.35517/n1.conf#
```

CASO 4: *Configure el firewall del Cliente de modo que, para cualquiera de los puntos anteriores, el mismo pueda establecer hacia el Servidor cualquier tipo de comunicación (siempre y cuando el Servidor se lo permita), pero sin permitir que el Web Server pueda iniciar comunicaciones nuevas hacia él*

- Configuración del nodo:
  - `iptables -A OUTPUT -d 10.0.0.10 -p tcp -j ACCEPT`
  - `iptables -A INPUT -s 10.0.0.10 -m state -state NEW -j DROP`
- Salida de `iptables -t nat -nL -v`

```
root@n2:/tmp/pycore.35517/n2.conf# iptables -nL -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      *       10.0.0.10            0.0.0.0/0           state NEW

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            10.0.0.10
root@n2:/tmp/pycore.35517/n2.conf#
```

7.2.2. Topología ruteada

Al igual que con la topología anterior, se utilizó el *script* “get-info.sh” para recuperar la información de cada nodo (los archivos de salida se pueden encontrar en “assets/iptables/output ruteada”).

```
root@n2:/tmp/pycore.38633/n2.conf# ping -c 3 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=63 time=0.104 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=63 time=0.108 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=63 time=0.087 ms

--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
rtt min/avg/max/mdev = 0.087/0.099/0.108/0.014 ms
```

Figura 4: Nodo *n2* haciendo `ping` al servidor *n1*

Modificando las reglas de `iptables` del router *n5* para que no deje pasar los paquetes que no salen de él ni tampoco son para él (`iptables -P FORWARD DROP`), los mensajes de `ping` de los nodos *n2* y *n3* no llegan a destino. Los mensajes vuelven a llegar a destino una vez removida la regla (`iptables -P FORWARD ACCEPT`).

El archivo "n1-con-servidores" ubicado en el directorio "assets/iptables/output-ruteada" contiene la salida de los comandos `netstat -nat` y `netstat -nau`, ejecutados en el nodo *n1* (el servidor).

Firewall con estados (*stateful*)

A continuación se indican las siguientes reglas a aplicar en el *router-firewall*.
Utilizar política restrictiva

- Configuración del *firewall*:
  - `iptables -P FORWARD DROP`
  - `iptables -P INPUT DROP`
  - `iptables -P OUTPUT DROP`
- Salida de `iptables -nL -v` en el *firewall*

```
root@n5:/tmp/pycore.42457/n5.conf# iptables -nL -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

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

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@n5:/tmp/pycore.42457/n5.conf#
```

Permitir únicamente el acceso desde la LAN a los servicios FTP, SSH, y HTTP que corren en el servidor

- Configuración del *firewall*:
  - `iptables -A FORWARD -s 10.0.0.0/24 -d 10.0.1.10 -p tcp -m multiport -dports 21,22,80 -j ACCEPT`
  - `iptables -A FORWARD -m state -state ESTABLISHED,RELATED -j ACCEPT`
- Salida de `iptables -nL -v` en el *firewall*

```
root@n5:/tmp/pycore.42457/n5.conf# iptables -nL -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       10.0.0.0/24          10.0.1.10           multiport dports 21,22,80
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@n5:/tmp/pycore.42457/n5.conf#
```

Ninguna otra comunicación hacia el router-firewall debe permitirse, ya sea desde la LAN como desde el Servidor.

- Al ser una política restrictiva, y aplicar la regla `INPUT DROP`, esta condición se cumple.

Desde el firewall se deben poder iniciar conexiones SSH y HTTPS al servidor

- Configuración del *firewall*:
  - `iptables -A OUTPUT -d 10.0.1.10 -p tcp -m multiport -dports 22,443 -j ACCEPT`
- Salida de `iptables -nL -v` en el *firewall*

```
root@n5:/tmp/pycore.42457/n5.conf# iptables -nL -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       10.0.0.0/24          10.0.1.10           multiport dports 21,22,80
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            10.0.1.10           multiport dports 22,443
root@n5:/tmp/pycore.42457/n5.conf#
```

Desde el servidor se debe permitir el acceso al servicio SSH de las PCs de la LAN

Las siguientes reglas se deben aplicar en el nodo *n1* (servidor):

- Configuración del servidor:
  - `iptables -A INPUT -s 10.0.0.0/24 -p tcp -dport 22 -j ACCEPT`
- Salida de `iptables -nL -v` en el servidor

```
root@n1:/tmp/pycore.42457/n1.conf# iptables -nL -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       10.0.0.0/24          0.0.0.0/0           tcp dpt:22

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

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@n1:/tmp/pycore.42457/n1.conf#
```

Firewall sin estados (*stateless*)

Configure el firewall con una política restrictiva y sin estados, de modo que se permita únicamente el acceso desde el cliente al servidor, a los servicios de FTP, SSH y HTTP

- Configuración del *firewall*:
  - `iptables -P INPUT DROP`
  - `iptables -P FORWARD DROP`
  - `iptables -P OUTPUT DROP`
  - `iptables -A FORWARD -s 10.0.0.0/24 -d 10.0.1.10 -p tcp -m multiport -dports 21,22,80 -j ACCEPT`
- Salida de `iptables -nL -v` en el *firewall*

```
root@n5:/tmp/pycore.42457/n5.conf# iptables -nL -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       10.0.0.0/24          10.0.1.10           multiport dports 21,22,80

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@n5:/tmp/pycore.42457/n5.conf#
```

Desde el firewall se deben poder iniciar conexiones SSH al servidor

- Configuración del *firewall*:
  - `iptables -A OUTPUT -d 10.0.1.10 -p tcp -dport 22 -j ACCEPT`
- Salida de `iptables -nL -v` en el *firewall*

```
root@n5:/tmp/pycore.42457/n5.conf# iptables -nL -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       10.0.0.0/24          10.0.1.10           multiport dports 21,22,80

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            10.0.1.10           tcp dpt:22
root@n5:/tmp/pycore.42457/n5.conf#
```

Idem punto anterior pero con una política permisiva

- Configuración del *firewall*:
  - `iptables -P INPUT ACCEPT`
  - `iptables -P FORWARD ACCETP`
  - `iptables -P OUTPUT ACCEPT`
  - `iptables -A FORWARD -s 10.0.0.0/24 -d 10.0.1.10 -p tcp -m multiport -dports 0:20,23:79,81:65535 -j DROP`
- Salida de `iptables -nL -v` en el *firewall*

```
root@n5:/tmp/pycore.42457/n5.conf# iptables -nL -v
Chain INPUT (policy ACCEPT 0 packets, 0 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
    0     0 DROP       tcp  --  *      *       10.0.0.0/24          10.0.1.10           multiport dports 0:20,23,79,81:65535

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
root@n5:/tmp/pycore.42457/n5.conf#
```

Configuración hogareña

Para configurar un *firewall* de modo que se permita que un cliente de la red puede realizar cualquier tipo de conexión (a servidores web) hacia fuera de la red (Internet), pero que desde Internet no puedan ingresar comunicaciones, se deben aplicar las siguientes reglas:

- `iptables -P FORWARD DROP` - política restrictiva
- `iptables -A FORWARD -s 192.168.0.0/24 -p tcp -dport 80 -j ACCEPT` - se deja pasar todo el tráfico interno que quiera ir hacia afuera de la red, a algún servidor web
- `iptables -A FORWARD -p tcp -sport 80 -d 192.168.0.0/24 -m state -state RELATED,ESTABLISHED -j ACCEPT` - el único tráfico que se deja ingresar a la red es el que está relacionado con alguna conexión previamente realizada

7.3. Comparación entre *stateful* y *stateless*

No se puede definir si es una alternativa es mejor que la otra tan sencillamente; la decisión de elegir uno u el otro depende mucho de la situación en la que se tenga que implementar el *firewall* y con qué recursos se cuente.

Los *firewall sin estado* (*stateless*) son más rápidos y funcionan más eficientemente (realizan el filtrado en base a campos estáticos de los paquetes, como pueden ser direcciones IP o puertos de origen y/o destino). Por su lado, los *firewall con estado* (*stateful*) son mejores para detectar conexiones no autorizadas o falsificadas, gracias a su capacidad de tener una visión general del estado de la red, y su habilidad para *contextualizar* el tráfico [4].

Referencias

[1] Fernando Catoira. *Pentesting: Fingerprinting para detectar sistema operativo*. Oct. de 2012. URL: https://www.welivesecurity.com/la-es/2012/10/18/pentesting-fingerprinting-para-detectar-sistema-operativo/.

[2] Nikos Danopoulos. *DNS Enumeration Techniques in Linux*. Nov. de 2016. URL: https://resources.infosecinstitute.com/dns-enumeration-techniques-in-linux/.

[3] Roberto C. González. *Enumeración*. Jun. de 2017. URL: http://ehack.info/enumeracion/.

[4] Lanner-America. *Stateless Vs. Stateful Packet Filtering Firewalls: Which is the Better?* Mayo de 2018. URL: https://www.lanner-america.com/blog/stateless-vs-stateful-packet-filtering-firewalls-better/.

[5] Netcraft LTD. *How does the Risk Rating work?* 2018. URL: https://toolbar.netcraft.com/help/faq/index.html#riskrating.

[6] OpenCampus. *What is Footprinting*. URL: https://www.greycampus.com/opencampus/ethical-hacking/what-is-footprinting.

[7] Victor Torres. *Footprinting (Reconocimiento)*. Mar. de 2012. URL: http://ciberinfosystem.blogspot.com/2012/03/footprinting-reconocimiento.html.