A1 – Exploraciones de red con Nmap y Nessus

Joaquín García Alfaro

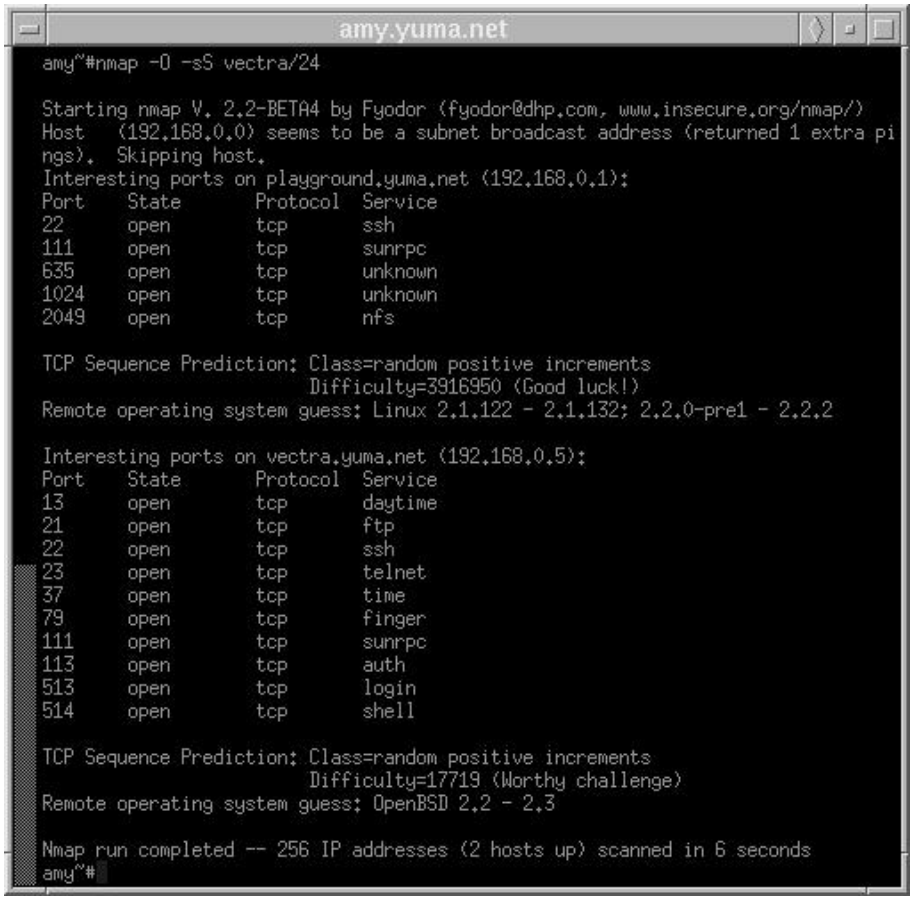
Índice

1.1. Nmap

La aplicación *Network Mapper*, más conocida como Nmap, es una de las herramientas más avanzadas para realizar exploración de puertos desde sistemas GNU/Linux. Nmap implementa la gran mayoría de técnicas conocidas para exploración de puertos y permite descubrir información de los servicios y sistemas encontrados, así como el reconocimiento de huellas identificativas de los sistemas escaneados. La siguiente imagen muestra un ejemplo de exploración de puertos mediante la herramienta *Nmap*:

A tener en cuenta

La mayor parte de herramientas de exploración de puertos pueden ser muy "ruidosas" y no son bien vistas por los administradores de red. Es altamente recomendable no utilizar estas herramientas sin el consentimiento explícito de los responsables de la red.



Aunque inicialmente Nmap fue pensada como una herramienta deshonesta para la realización de ataques, actualmente es una de las aplicaciones más utilizadas por administradores de red para la realización de comprobaciones de seguridad. Entre las muchas posibilidades que Nmap nos ofrece podríamos destacar las siguientes:

- **Auditorías de nuestra red.** Nmap permite una rápida visualización de puertos inseguros o abiertos por equivocación en los equipos de nuestra red.

- **Comprobación de la configuración de los elementos de seguridad.** Nmap puede ser de gran utilidad para comprobar la configuración de los elementos de seguridad de nuestra red como, por ejemplo, sistema cortafuegos, sistemas de detección de intrusos, etc. Por ejemplo, la realización de una exploración de puertos mediante Nmap desde el exterior de nuestra red podría asegurarnos que nuestro sistema cortafuegos está realizando el bloqueo de paquetes de forma correcta.

- **Comprobación de la configuración de los elementos de red.** Mediante Nmap podemos realizar una serie de comprobaciones de los dispositivos de conectividad y encaminamiento de nuestra red y detectar así si hay algún malfuncionamiento o, al contrario, si todo funciona con normalidad.

Desde el punto de vista de herramienta para la exploración de equipos de red, Nmap es más que un simple escáner de puertos. Algunas de las funcionalidades más interesantes que han hecho de Nmap una de de las herramientas de exploración más importantes y populares son las siguientes:

- **Alta velocidad de exploración.** Nmap ofrece una extraordinaria velocidad a la hora de realizar una comprobación de sistemas activos y servicios ofrecidos por dichos sistemas.

- **Descubrimiento de huellas identificativas.** Nmap ofrece la posibilidad de detectar la mayor parte de sistemas existentes en la actualidad con un alto grado de fiabilidad. Aunque Nmap no hace más que una predicción del sistema operativo que se esconde detrás del equipo que está explorando, dicha predicción se basa en contrastar la información recibida frente a una gran base de datos de respuestas basadas en tráfico IP, ICMP, UDP y TCP de centenares de sistemas operativos existentes en la actualidad.

- **Predicción de números de secuencia.** Todo el tráfico basado en protocolo TCP requiere un patrón aleatorio para establecer un inicio de conexión con el sistema remoto. Dicho patrón será establecido durante el protocolo de conexión de tres pasos de TCP mediante la utilización de números de secuencia aleatorios. En algunos sistemas operativos puede ser que estos números de secuencia no presenten un índice de aleatoriedad elevado, por lo que es posible realizar una predicción del número de secuencia que se utilizará en la siguiente conexión y realizar, por ejemplo, un secuestro de conexión TCP. Nmap ofrece la posibilidad de poder realizar una predicción de cómo se comporta la aleatoriedad de los números de secuencia del equipo al que está explorando.

- **Habilidad para imitar distintos aspectos de una conexión TCP.** El establecimiento de una conexión TCP requiere cierto periodo de tiempo (del orden de milisegundos). Muchos sistemas cortafuegos pueden estar configurados para descartar el primer paquete TCP de este establecimiento de sesión (con la bandera de TCP/SYN activada), y evitar que desde el exterior un atacante pueda establecer una conexión contra los equipos del sistema a proteger. La mayoría de los exploradores de puertos tradicionales utilizan este paquete de TCP/SYN para realizar este tipo de exploraciones, por lo que el sistema cortafuegos podrá bloquear dichos intentos de conexión y evitar que el explorador de puertos los detecte como activos. Nmap, sin embargo, es capaz de generar

paquetes TCP que atraviesen esta protección ofrecida por los sistemas cortafuegos y ofrecer una lista de los servicios activos en el equipo de destino.

- **Funciones para realizar suplantación.** Gracias a la capacidad de suplantación (*spoofing*) que ofrece Nmap, es posible que un atacante pueda hacerse pasar por otros equipos de confianza con el objetivo de atravesar la protección que ofrece el sistema cortafuegos de una red. Utilizado de forma correcta por parte de los administradores de dicha red, Nmap puede ayudar a modificar la configuración de estos sistemas cortafuegos disminuyendo, así, la posibilidad de recibir un ataque de suplantación.

- **Habilidad para controlar la velocidad de exploración.** La mayoría de los sistemas de detección de intrusos actuales suelen generar alertas en el momento en que detectan la presencia de una exploración secuencial de puertos contra la red que están protegiendo. De este modo, el sistema de detección podría avisar ante la presencia de diferentes exploraciones contra los equipos de dicha red. Mediante Nmap es posible modificar el comportamiento de la exploración para evitar así la detección por parte de sistemas de detección de intrusos tradicionales. De nuevo, una utilización correcta de Nmap por parte de los administradores de una red facilitará la adecuada configuración de los sistemas de detección para no dejar pasar desapercibidos este tipo de exploraciones.

- **Posibilidad de guardar y leer ficheros de texto.** Nmap ofrece la posibilidad de guardar sus resultados en forma de ficheros de texto de salida, de forma que otras aplicaciones puedan leer estos resultados, así como la posibilidad de leer la información de exploraciones anteriores por parte del propio Nmap.

1.1.1. Opciones de exploración de Nmap

Una de las posibilidades más interesantes de Nmap es su versatilidad a la hora de realizar sus exploraciones. Nmap puede ser utilizado tanto para una sencilla exploración rutinaria contra los equipos de nuestra red, como para realizar una compleja predicción de números de secuencia y descubrimiento de la huella identificativa de sistemas remotos protegidos por sistemas cortafuegos. Por otro lado, puede ser utilizado para la realización de una única exploración o de forma interactiva, para poder realizar multiples exploraciones desde un mismo equipo.

En general, como la mayoría de aplicaciones ejecutadas desde consola, Nmap puede combinar toda una serie de opciones de exploración que tienen sentido en conjunto, aunque también existen otras operaciones que son especificas para ciertos modos de exploración. A la hora de lanzar Nmap con múltiples opciones a la vez, la aplicación tratará de detectar y advertirnos sobre el uso de combinaciones de opciones incompatibles o no permitidas.

A continuación, mostramos algunas de las opciones de configuración de Nmap más básicas para la realización de exploraciones:

- **-P0** Por defecto, Nmap envía un mensaje ICMP de tipo `echo` a cada equipo que va a

explorar. Activando esta opción deshabilitaremos este comportamiento. Esta opción suele ser de gran utilidad si la exploración a realizar se realiza contra sistemas que aparentemente no han de estar activos y que, por tanto, no responderán a este primer mensaje ICMP enviado por Nmap. Por contra, si utilizamos esta información, deberemos ser conscientes que la información proporcionada por Nmap puede no ser del todo precisa.

- **-PT** Indica a Nmap que utilize paquetes TCP en lugar de mensajes ICMP para la realización del *ping* inicial contra el objetivo a explorar. Para ello, Nmap enviará un paquete del tipo TCP/ACK y esperará el envío de una respuesta TCP/RST por parte del equipo remoto. Esta opción es bastante interesante, ya que nos permite comprobar la existencia de un sistema cortafuegos entre el equipo donde estamos realizando la exploración y el equipo remoto. Aunque muchos cortafuegos filtran el tráfico ICMP de tipo `echo` para evitar la realización de `pings` desde el exterior, la mayoría suele permitir la entrada y salida de paquetes TCP/ACK y TCP/RST.

- **-v** Si activamos la opción *verbose* al lanzar Nmap, la aplicación nos irá mostrando las respuestas de los equipos explorados a medida que la vaya recibiendo. Si activamos dos veces la opción (`nmap -v -v`), recibiremos información adicional, dependiendo del tipo de exploración realizada.

- **-O** Esta opción indica a Nmap que trate de hacer una predicción del sistema operativo que se está ejecutando tras el equipo o los equipos que están siendo explorados. Esta es una opción muy apreciada por un posible atacante, ya que le permitirá iniciar una búsqueda de herramientas de explotación específicas, según el sistema operativo que se encuentre tras los equipos explorados.

  Como ya hemos adelantado anteriormente, Nmap utiliza una base de datos de huellas identificativas de la mayor parte de sistemas operativos existentes para apurar al máximo la precisión de sus resultados. Además, los desarrolladores de Nmap tratan de mantener dicha base de datos lo más actualizada posible para poder estar seguros de la eficiencia de predicción de Nmap. Gran parte de la información almacenada en esta base de datos se refiere a las peculiaridades de implementación de la pila TCP/IP de los distintos sistemas operativos existentes.

Opciones `-s` de Nmap

Todas aquellas opciones de Nmap precedidas por el prefijo `-s` se consideran opciones para realizar exploraciones silenciosas (pero detectables igual que cualquier exploración de puertos).

- **-sS** Utiliza una exploración de puertos TCP silencios basada en el envío de paquetes TCP/SYN. Si bien Nmap no finaliza el protocolo de conexión de forma expresa, Nmap es capaz de recoger la información suficiente para detectar todos aquellos servicios TCP ofrecidos por el equipo remoto.

- **-sP** Al activar esta opción, Nmap utilizará únicamente mensajes ICMP para la realización de la exploración.

1.1.2. Algunos ejemplos sencillos de exploración con Nmap

Como primer ejemplo de exploración con Nmap supondremos que, como administradores de nuestra red, queremos realizar únicamente una exploración mediante el uso de mensajes ICMP de tipo `echo`. El objetivo de dicha exploración podría ser la comprobación de los equipos de nuestra red local que están activos. La red local de nuestro ejemplo corresponde a la dirección IP 10.0.1.100/30. Para ello, lanzaremos Nmap con la opción `-sP` contra la dirección `10.0.1.100/30`. Si, adicionalmente, añadimos la opción `-v` para recibir la información de la exploración tan pronto llegue, éste sería el resultado obtenido de dicha exploración:

```
root$nmap -v -sP 10.0.1.100/30

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-02-01 13:28 CET
Host 10.0.1.100 appears to be up.
Host 10.0.1.101 appears to be down.
Host 10.0.1.102 appears to be down.
Host 10.0.1.103 appears to be down.
Nmap run completed -- 4 IP addresses (1 host up) scanned in 1.046 seconds
```

A continuación, podríamos realizar una exploración de los servicios abiertos en el equipo que acabamos de ver activo y almacenar los resultados obtenidos, con la siguiente combinación de opciones de nmap (la opción `-oN` es utilizada para almacenar la información reportada por nmap en un fichero de *log*):

```
root$nmap -P0 -oN output.txt 10.0.1.100

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-02-01 13:43 CET
Interesting ports on 10.0.1.100:
(The 1644 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
13/tcp   open  daytime
19/tcp   open  chargen
22/tcp   open  ssh
25/tcp   open  smtp
37/tcp   open  time
80/tcp   open  http
111/tcp  open  rpcbind
723/tcp  open  omfs
5801/tcp open  vnc-http-1
5901/tcp open  vnc-1
6000/tcp open  X11
6001/tcp open  X11:1

Nmap run completed -- 1 IP address (1 host up) scanned in 1.719 seconds
```

```
root$cat output.txt
# nmap 3.48 scan initiated Sun Feb  1 13:43:00 2004 as: nmap -P0 -oN output.txt
10.0.1.100
Interesting ports on 10.0.1.100:
(The 1644 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
13/tcp   open  daytime
19/tcp   open  chargen
22/tcp   open  ssh
25/tcp   open  smtp
37/tcp   open  time
80/tcp   open  http
111/tcp  open  rpcbind
723/tcp  open  omfs
5801/tcp open  vnc-http-1
5901/tcp open  vnc-1
6000/tcp open  X11
6001/tcp open  X11:1

# Nmap run completed at Sun Feb  1 13:43:02 2004 -- 1 IP address (1 host up)
scanned in 1.719 seconds
```

Si quisieramos ahora realizar una exploración selectiva contra el equipo 10.0.1.100, tratando de descubrir únicamente si están activos los servicios `ssh` y `web` de dicho equipo, podríamos realizar la exploración con las siguientes opciones de nmap (de nuevo, utilizando la opción `-oN` para almacenar la información reportada en un fichero de *log*):

```
root$nmap -sX -p 22,80 -oN output.txt 10.0.1.100

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-02-01 13:55 CET
Interesting ports on 10.0.1.100:
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap run completed -- 1 IP address (1 host up) scanned in 1.181 seconds
```

Al activar la opción `-sX`, Nmap realizará una exploración silenciosa contra el equipo `10.0.1.100` mediante la técnica de exploración *Xmas Tree**. No todos los sistemas operativos contestarán correctamente a una exploración de puertos utilizando la técnica de *Xmas Tree*, dado que algunos como, por ejemplo, OpenBSD, HP/UX, IRIX, Microsoft Windows, etc. no siguen el estándar propuesto por las RFCs del IETF.

Xmas Tree

Al igual que una exploración *TCP FIN*, la técnica de la exploración *TCP Xmas Tree* enviará un paquete `FIN`, `URG` y `PUSH` a un puerto, y esperará respuesta. Si se obtiene como resultado un paquete de reset, significa que el puerto está cerrado.

1.1.3. Utilización de Nmap en modo interactivo

Nmap ofrece un modo de trabajo interactivo que permite a los administradores de red la realización de multiples opciones de exploración desde una única sesión de consola. Para ello, tan sólo deberemos lanzar Nmap con la opción `--interactive` activada. A partir de ese momento, nos aparecerá como prefijo de consola la cadena `nmap>` desde la que podremos ir ejecutando las diferentes opciones de Nmap a nivel de comandos.

En la siguiente figura podemos ver una sesión de Nmap interactiva desde la cual hemos realizado una búsqueda de equipos activos en nuestra red local y, posteriormente, una exploración de servicios contra los equipos activos:

```
root$nmap --interactive

Starting nmap V. 3.48 ( http://www.insecure.org/nmap/ )
Welcome to Interactive Mode -- press h <enter> for help
nmap> n -v -sP 10.0.1.100/30
Host 10.0.1.100 appears to be up.
Host 10.0.1.101 appears to be down.
Host 10.0.1.102 appears to be down.
Host 10.0.1.103 appears to be down.
Nmap run completed -- 4 IP addresses (1 host up) scanned in 1.290 seconds
nmap> n -P0 10.0.1.100
Interesting ports on 10.0.1.100:
(The 1644 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
13/tcp   open  daytime
19/tcp   open  chargen
22/tcp   open  ssh
25/tcp   open  smtp
37/tcp   open  time
80/tcp   open  http
111/tcp  open  rpcbind
723/tcp  open  omfs
5801/tcp open  vnc-http-1
5901/tcp open  vnc-1
6000/tcp open  X11
6001/tcp open  X11:1

Nmap run completed -- 1 IP address (1 host up) scanned in 1.516 seconds
nmap> quit
Quitting by request.
root$
```

Si nos fijamos en el ejemplo anterior, los pasos seguidos son los siguientes:

1) Entramos en una nueva sesión interactiva de Nmap:

```
nmap --interactive
```

2) Utilizamos el comando `n` para realizar una nueva exploración de Nmap pasándole como argumentos las opciones necesarias para realizar una búsqueda de equipos activos en la red 10.0.1.100/30:

```
nmap> n -v -sP 10.0.1.100/30
```

3) Utilizamos de nuevo el comando `n` para realizar una exploración de puertos TCP abiertos en el equipo 10.0.1.100:

```
nmap> n -P0 10.0.1.100
```

4) Salimos de la sesión interactiva con Nmap mediante el comando `quit`:

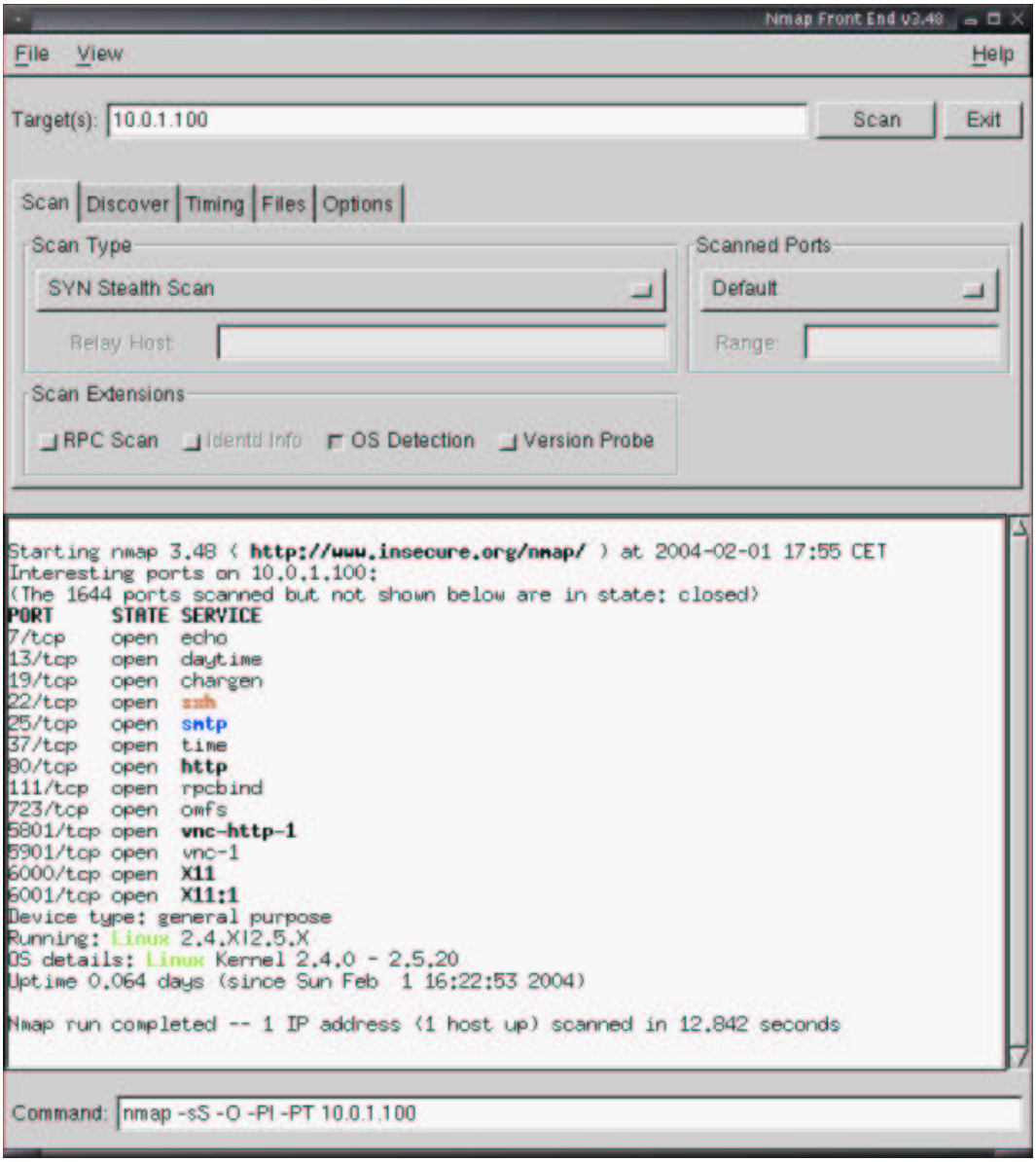
```
nmap> quit
```

Para ver los diferentes comandos que Nmap nos ofrece desde su modo interactivo, entraremos el comando `help` desde el guión de comando `nmap>`:

```
nmap> help
Nmap Interactive Commands:
n <nmap args> -- executes an nmap scan using the arguments given and
waits for nmap to finish.  Results are printed to the
screen (of course you can still use file output commands).
! <command>   -- runs shell command given in the foreground
x             -- Exit Nmap
f [--spoof <fakeargs>] [--nmap_path <path>] <nmap args>
-- Executes nmap in the background (results are NOT
printed to the screen).  You should generally specify a
file for results (with -oX, -oG, or -oN).  If you specify
fakeargs with --spoof, Nmap will try to make those
appear in ps listings.  If you wish to execute a special
version of Nmap, specify --nmap_path.
n -h          -- Obtain help with Nmap syntax
h             -- Prints this help screen.
Examples:
n -sS -O -v example.com/24
f --spoof "/usr/local/bin/pico -z hello.c" -sS -oN e.log example.com/24
```

1.1.4. Utilización de NmapFE como interfaz gráfica de Nmap

Por último, cabe destacar la posibilidad de trabajar de forma gráfica con Nmap a través del *front-end NmapFE*. Al igual que Nmap, y muchas otras herramientas relacionadas con Nmap, esta interfaz gráfica puede ser descargada del sitio web `www.insecure.org`. La siguiente figura muestra una sesión de exploración con Nmap desde esta interfaz gráfica.



1.2. Nessus

La herramienta Nmap que hemos visto en el apartado anterior es utilizada internamente por otras aplicaciones como, por ejemplo, escáners de vulnerabilidades, herramientas de detección de sistemas activos, servicios web que ofrecen exploración de puertos, etc.

Nmap, junto con **Nessus**, son dos de las herramientas más frecuentemente utilizadas tanto por administradores de redes, como por posibles atacantes, puesto que ofrecen la mayor parte de los datos necesarios para estudiar el comportamiento de un sistema o red que se quiere atacar.

Éste es el caso de la utilidad *Nessus*, una utilidad para comprobar si un sistema es vulnerable a un conjunto muy amplio de problemas de seguridad almacenados en su base de datos. Si encuentra alguna de estas debilidades en el sistema analizado, se encargará de informar sobre su existencia y sobre posibles soluciones.

Nessus es una herramienta basada en un modelo cliente-servidor que cuenta con su propio protocolo de comunicación. De forma similar a otros escáners de vulnerabilidades existentes, el trabajo correspondiente para explorar y probar ataques contra objetivos es realizado por el servidor de Nessus (`nessusd`), mientras que las tareas de control, generación de informes y presentación de los datos son gestionadas por el cliente (`nessus`).

Así pues, Nessus nos permitirá una exploración proactiva de los equipos de nuestra red en busca de aquellas deficiencias de seguridad relacionada con los servicios remotos que ofrecen dichos equipos. La mayor parte de las alertas que Nessus reportará estarán relacionadas con las siguientes deficiencias:

- Utilización de servidores (o *daemons*) no actualizados y que presentan deficiencias de seguridad conocidas como, por ejemplo, versiones antiguas de *sendmail*, *Finger*, wu-ftpd, etc.

- Deficiencias de seguridad relacionadas con una mala configuración de los servidores como, por ejemplo, permisos de escritura para usuarios anónimos por parte de un servidor de ftp.

- Deficiencias de seguridad relacionadas con la implementación de la pila TCP/IP del equipo remoto.

- Deficiencias de seguridad relacionadas con servidores de X Windows mal configurados o que presentan vulnerabilidades de seguridad.

- Utilización de aplicaciones CGI desde servidores web mal configuradas o mal programadas y que suponen una brecha de seguridad contra el sistema que las alberga.

- Instalación de puertas traseras, troyanos, demonios de DDoS u otros servicios extraños en sistemas de producción.

Como veremos más adelante, Nessus se compone de un conjunto de *plug-ins* que realizarán varias simulaciones de ataque. Alguno de estos ataques *simulados* pueden llegar a ser peligrosos contra el sistema analizado. Aunque Nessus no podrá nunca llegar a destruir información del sistema remoto, sus acciones podrían:

- Conducir a una denegación de servicio del sistema remoto. Al hacer las comprobaciones necesarias para realizar el test de análisis, ciertos *plug-ins* de Nessus pueden hacer reaccionar el sistema remoto de forma inadecuada y hacer que éste falle.

- Generar una cantidad masiva de tráfico basura en la red, pudiendo llegar a afectar al trabajo normal de los usuarios de la red.

Por estos motivos, es importante conocer correctamente las distintas posibilidades de configuración de esta herramienta y las distintas opciones que nos ofrece. Por otro lado, es conveniente realizar las pruebas de Nessus en horarios programados, con baja carga de trabajo en los equipos analizados. Es importante, por ejemplo, estar seguros de que los equipos que van a ser analizados pueden ser reiniciados, en caso de problemas, sin que esto afecte a ningún usuario legítimo de la red. También es importante tener presente que la red que va a ser analizada puede estar durante un breve periodo de tiempo saturada y que, por tanto, no estamos afectando a la producción normal de los servicios de dicha red.

1.2.1. Relación entre cliente y servidor de Nessus

Como ya hemos comentado anteriormente, para la elaboración de un escáner de vulnerabilidades, Nessus consta de dos aplicaciones. Por un lado, un servidor (`nessusd`) que será ejecutado en la máquina desde donde partirá el escaneo, y un cliente (`nessus`), que se comunicará a través de *sockets* al servidor. Generalmente, cliente y servidor se estarán ejecutando en distintas máquinas. Mientras que el cliente de nessus consta de una interfaz gráfica de usuario, el servidor de Nessus es una aplicación de consola que será ejecutada en modo *daemon*. En sistemas GNU/Linux, dependiendo de la distribución utilizada, el servidor de Nessus será ejecutado en modo *daemon* en el momento de iniciar el equipo por el guión de inicio del sistema correspondiente (generalmente situado en `/etc/rc.d/init.d/nessus`).

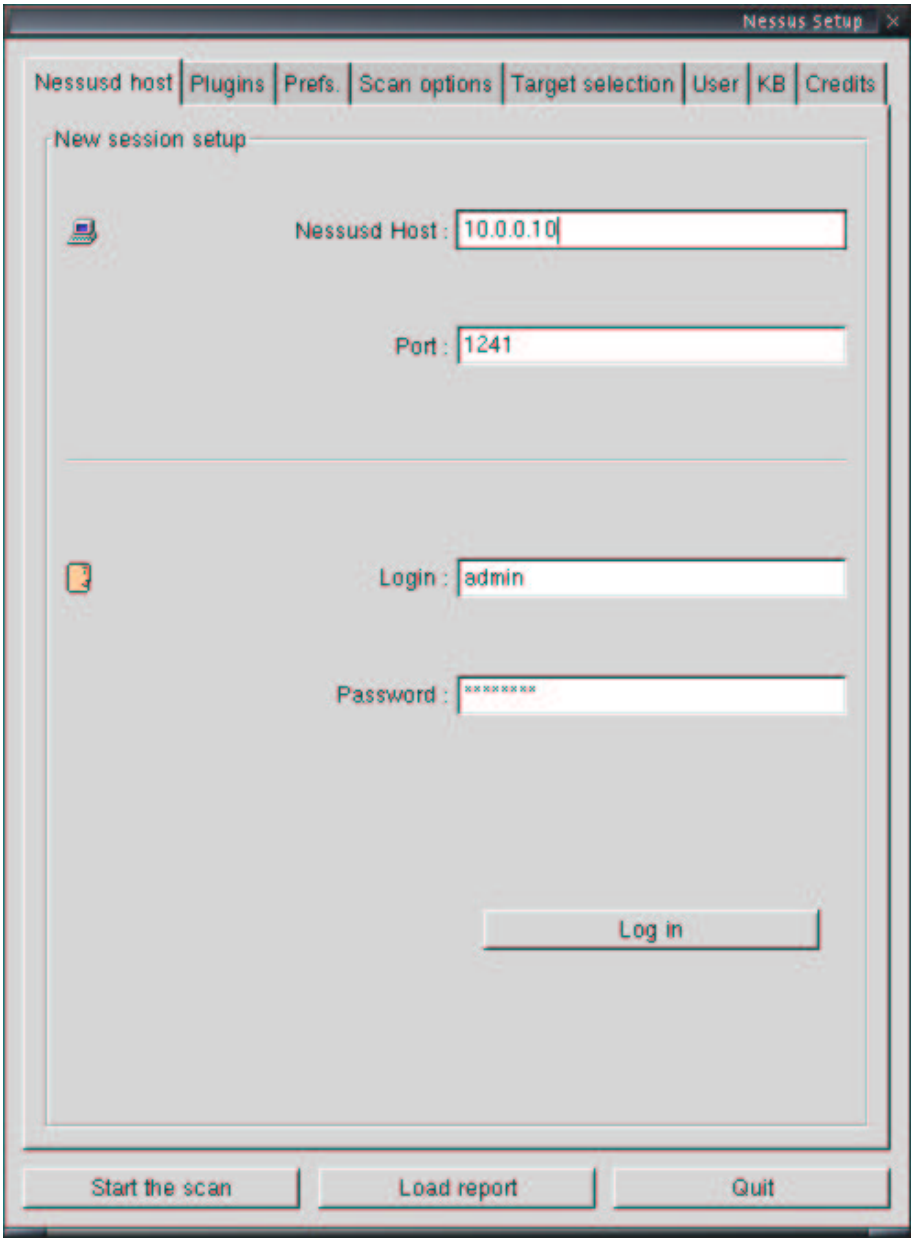
Clientes de Nessus

Existen distintos clientes de Nessus para conectarse a un servidor de Nessus, incluyendo clientes para sistemas Unix/Linux, Windows y Macintosh.

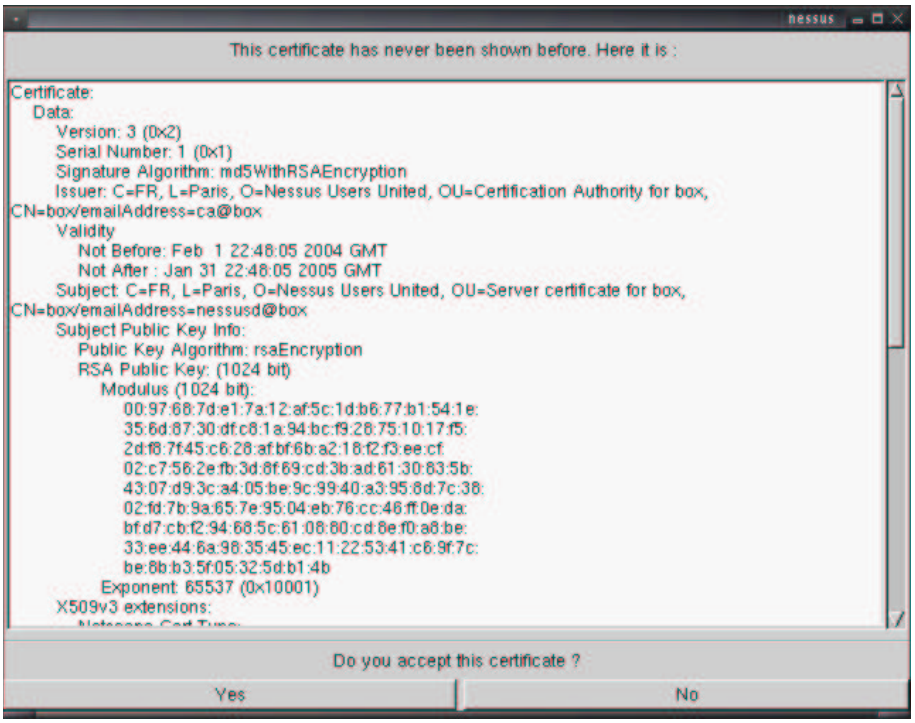
A la hora de conectarse al servidor, el cliente de Nessus realizará un proceso de autenticación. Este proceso de autenticación puede realizarse mediante el cifrado de los datos de autenticación o sin él. De las dos opciones anteriores, será preferible usar la versión cifrada, pues dificultará que un usuario ilegítimo pueda llegar a usar nuestro servidor de Nessus para realizar un análisis no autorizado.

En la siguiente figura podemos ver un cliente de Nessus ejecutado en un sistema GNU/Linux que ha utilizado el nombre de usuario `admin` para conectarse al servidor de Nessus que

está escuchando por el puerto `1241` del equipo con dirección IP `10.0.0.10`. Antes de poder realizar cualquier interacción con el servidor de Nessus que se está ejecutando en el equipo `10.0.0.10`, ha sido necesario realizar el proceso de autenticación entre ambos.



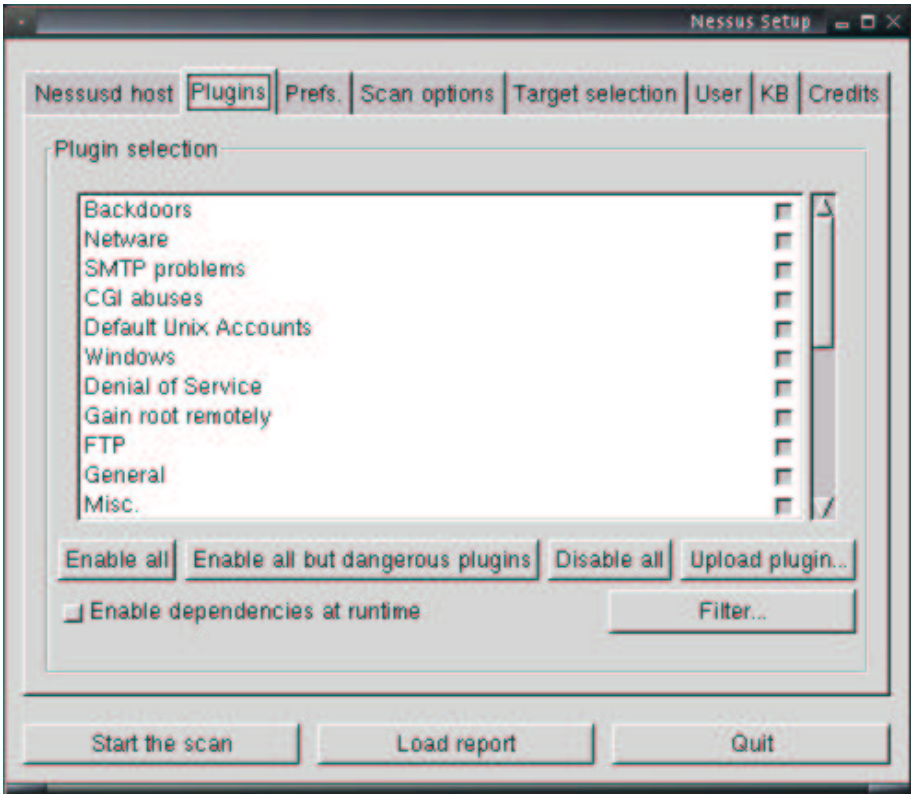
Si el mecanismo de autenticación escogido es mediante cifrado, el servidor de Nessus enviará un certificado digital con la información necesaria para poder realizar el proceso de autenticación correctamente. En la siguiente figura podemos ver un ejemplo de certificado enviado por el servidor de Nessus hacia el cliente.



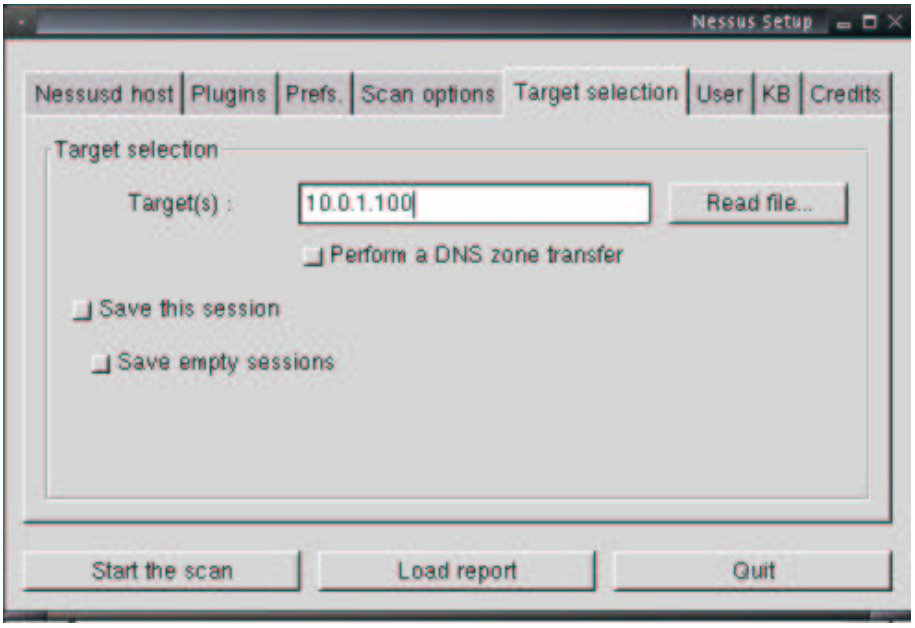
Es importante tener presente que, aparte del usuario remoto que utilizaremos para conectarnos al servidor de Nessus, existe también un usuario para poder utilizar el cliente de Nessus en la máquina local. Este usuario, utilizado para conectarse desde un equipo local al cliente de Nessus, será creado la primera vez que lancemos el cliente. Así, el cliente de Nessus solicitará un nombre de usuario y una contraseña local, para evitar que un usuario ilegítimo pueda utilizar el cliente de Nessus de forma no autorizada. Este nombre de usuario y su correspondiente contraseña no tienen nada que ver con el nombre de usuario y el mecanismo de autenticación que utilizaremos para la conexión remota contra el servidor de Nessus.

Una vez lanzado el cliente, será necesario proporcionar la dirección IP del equipo remoto donde se está ejecutando el servidor de Nessus, el puerto TCP por el que está escuchando, así como el nombre de usuario y la contraseña (o la clave correspondiente a la autenticación cifrada) asociada al servidor. Después de realizar el `Login`, la conexión con el servidor remoto de Nessus deberá iniciarse. Para ello, deberemos haber creado la correspondiente cuenta de usuario en el equipo remoto donde se encuentra el servidor. En caso contrario, el proceso de autenticación fallará.

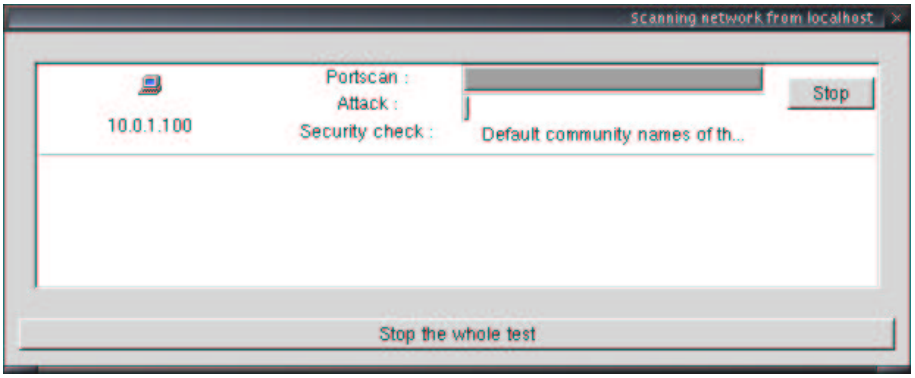
Si el proceso de autenticación es correcto y el cliente ha podido conectarse al servidor de Nessus, deberá mostrarse en la pantalla del cliente la lista de *plug-ins*. Desde esta pantalla podremos activar y desactivar los distintos *plug-ins* que el servidor utilizará durante el análisis. Si el proceso de autenticación no se ha realizado con éxito, el menu de *plug-ins* estará vacío. En la siguiente figura podemos ver la pantalla de *plug-ins* tras realizar un proceso de autenticación correcto.



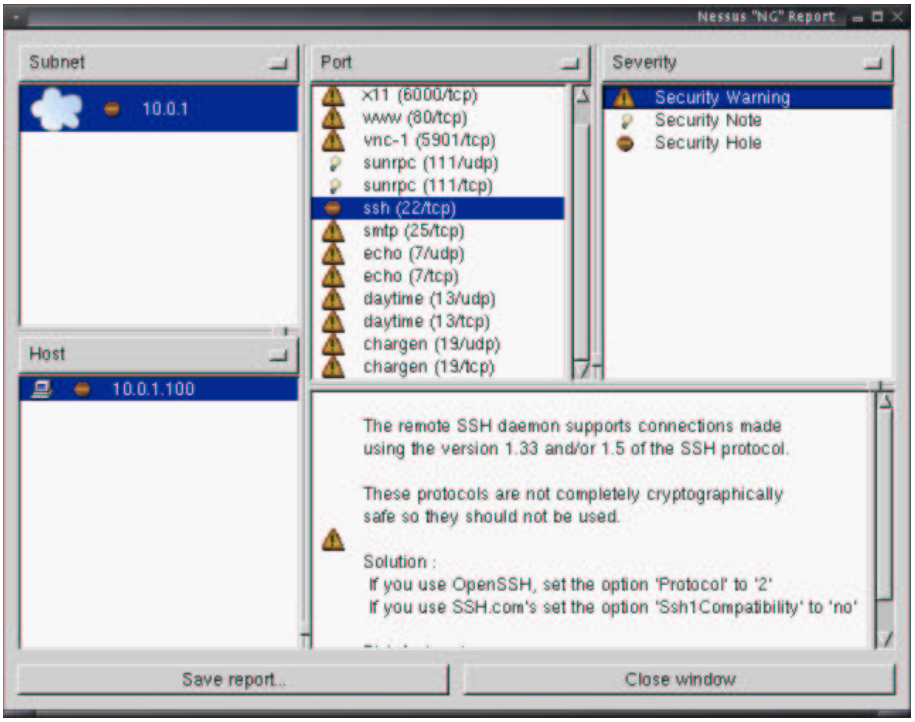
Una vez realizada la conexión con el servidor de Nessus, y habiendo seleccionado los *plug-ins* correspondientes a los análisis que deseamos realizar, podemos escoger el equipo (o una lista de equipos) a analizar. En la siguiente figura vemos, por ejemplo, cómo seleccionar desde el cliente de Nessus el equipo `10.0.1.100`. Hay que recordar que dicho equipo recibirá las diferentes pruebas desde la máquina donde se encuentra instalado el servidor de Nessus, no el cliente.



Tras seleccionar el objetivo, podemos iniciar el análisis mediante la opción correspondiente en la siguiente pantalla del cliente de Nessus. A medida que el análisis se vaya realizando, iremos viendo por pantalla las distintas operaciones que el servidor de Nessus va realizando contra el equipo o equipos seleccionados. La siguiente figura muestra el análisis de vulnerabilidades contra el equipo `10.0.1.100` seleccionado anteriormente.

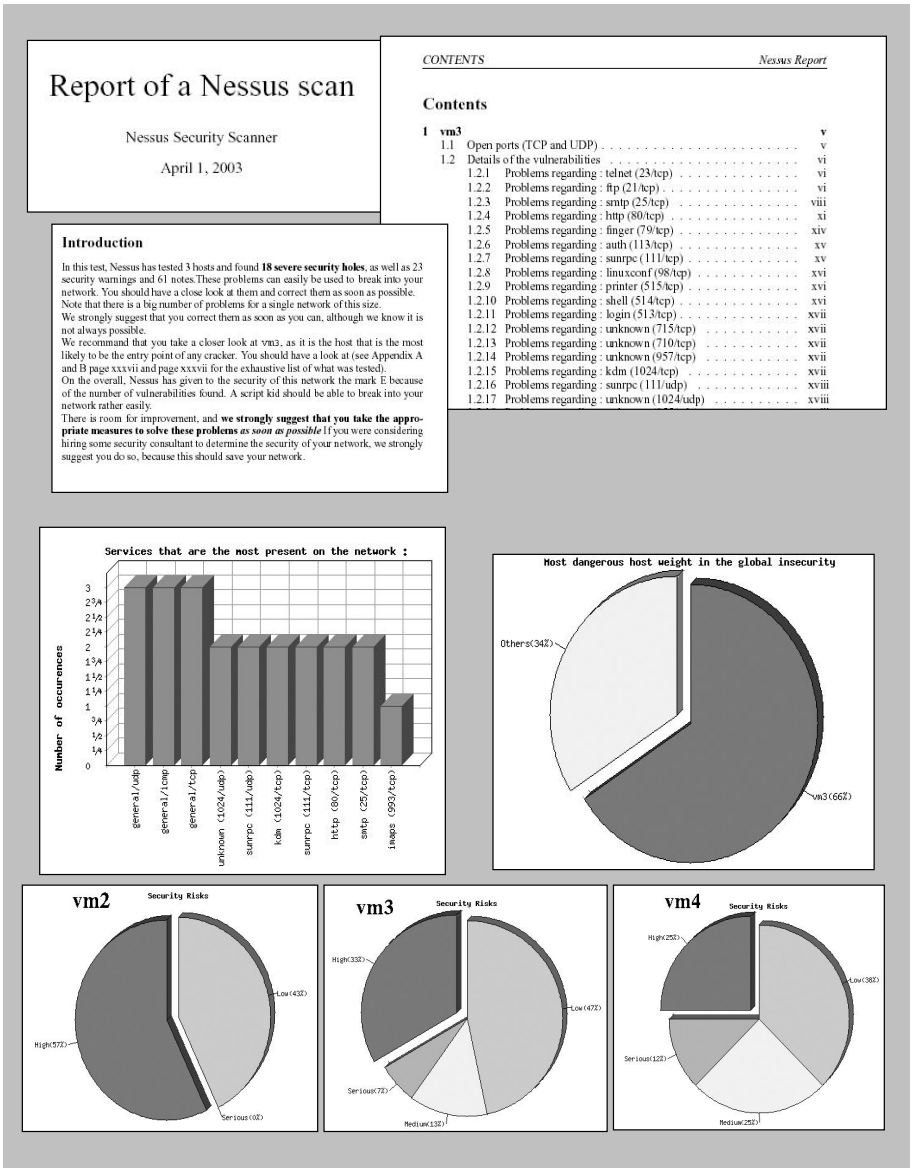


Una vez finalizada la exploración de vulnerabilidades por parte del servidor de Nessus, el cliente nos mostrará por pantalla los resultados de dicha exploración. Estos resultados dependerán de los *plug-ins* que hayamos activado, de las diferentes opciones de configuración del servidor, etc. Como vemos en la siguiente figura, la interfaz ofrecida por el cliente para mostrar los resultados permitirá su visualización por equipos, por redes, según la severidad de las vulnerabilidades detectadas, etc.



La aplicación también permitirá almacenar los resultados obtenidos durante la realización de la exploración en forma de informes. Las actuales más recientes del cliente de Nessus

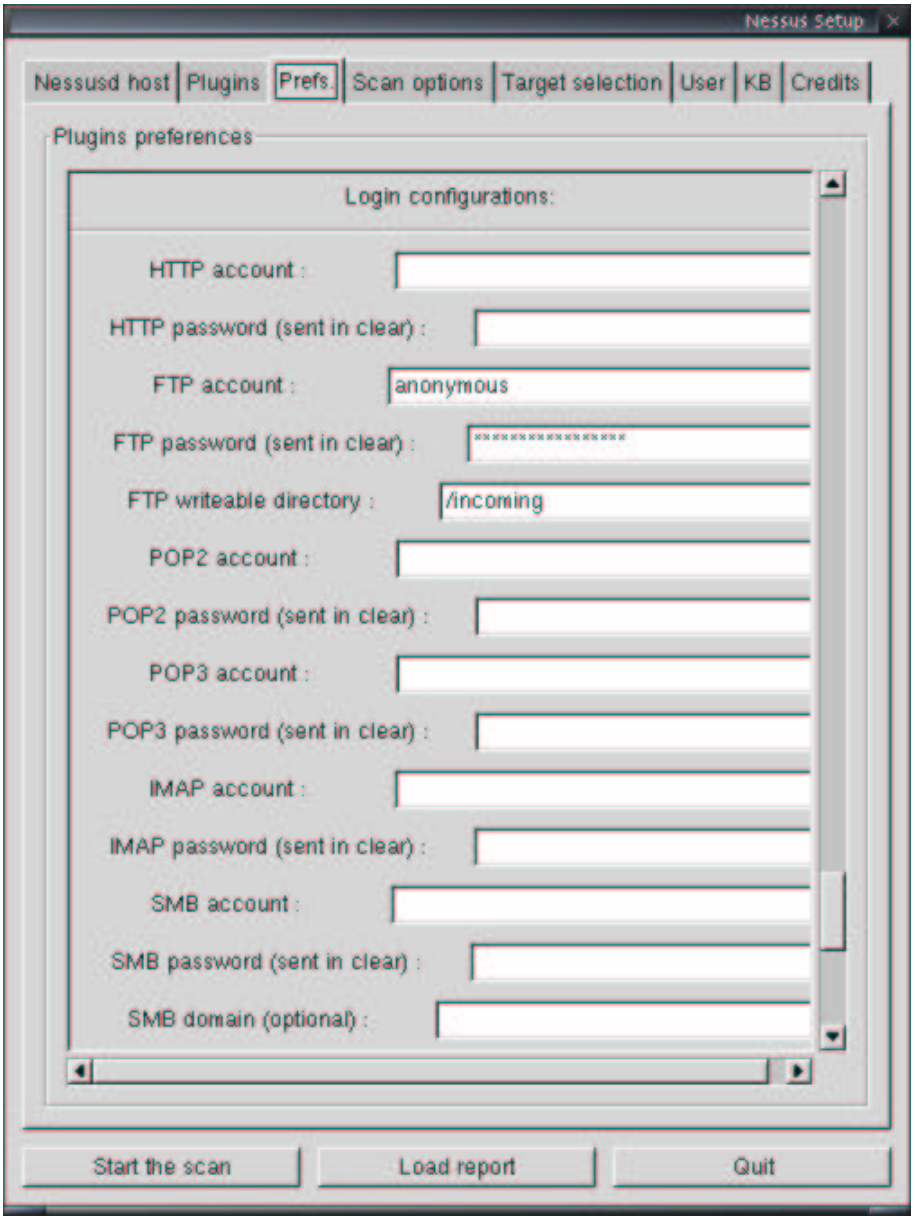
ofrecen un amplio rango de formatos para poder almacenar tales informes. Desde guardar el informe en un formato propio de la aplicación (para poder ser visualizado de nuevo desde otro cliente de Nessus) hasta la realización de completos informes en HTML, XML, LaTeX, etc. En la siguiente figura, podemos ver un ejemplo de informe generado mediante el cliente de Nessus en HTML y PDF.



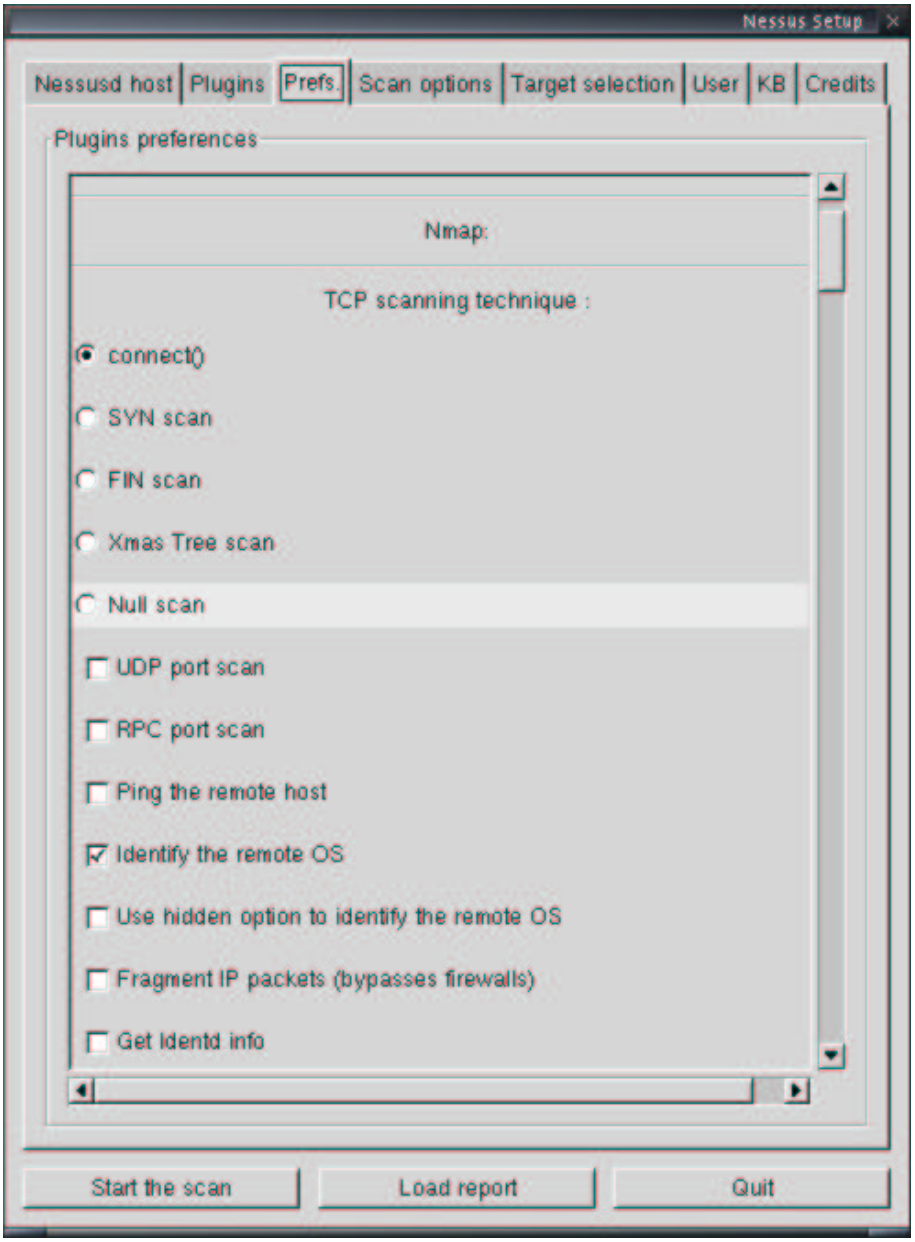
1.2.1.1. Configuración de *plug-ins*

Aunque la mayoría de los *plug-ins* de Nessus pueden ser utilizados sin necesidad de ajustes, otros *plug-ins* requerirán la inserción de información adicional para funcionar correctamente. Por ejemplo, los análisis relacionados con *plug-ins* de `ftp`, `smtp` y otros servicios similares con autenticación de usuarios requerirán la inserción de información relacionada con autenticación de usuarios. También es posible, por ejemplo, configurar el *plug-in* de `ftp` para que trate de almacenar información en el caso de encontrar una mala configuración en los permisos de escritura de los recursos asociados. Otra posibilidad es la

modificación de los parámetros de Nmap (relacionado con la exploración de puertos que el servidor de Nessus realizará). En la siguiente figura podemos ver la interfaz ofrecida por el cliente de Nessus para realizar algunas de las modificaciones comentadas.



La mayor parte de estas opciones estarán disponibles desde el cliente de Nessus en el momento de realizar la conexión con el correspondiente servidor. Aun así, una vez realizadas las modificaciones, éstas pueden ser almacenadas localmente para tratar de aplicarlas a servidores de Nessus adicionales. Así, si modificamos, por ejemplo, las preferencias de la exploración de puertos para que Nmap utilize UDP como protocolo, en lugar de TCP, podemos tratar de almacenar estas preferencias para que se apliquen de nuevo en futuras conexiones al mismo o a distintos servidores de Nessus. En la siguiente figura podemos ver la interfaz que ofrece el cliente de Nessus en cuanto a las opciones de exploración relacionadas con Nmap.



Cabe destacar que la actualización de *plug-ins* puede ser automatizada a través de guiones de sistema como, por ejemplo, los guiones del servicio `cron` de sistemas Unix. A medida que van apareciendo nuevas vulnerabilidades o problemas de seguridad en general, la comunidad de desarrollo de Nessus, así como administradores de red u otros profesionales en seguridad de redes, suelen traducir tales vulnerabilidades en forma de nuevos *plug-ins* de Nessus para ser descargados por parte de los usuarios de la aplicación. Aunque existen distintas formas de realizar el proceso de actualización, la más intuitiva consiste en una simple descarga desde el sitio web de Nessus con todo el paquete de *plug-ins*, y reemplazar los anteriores.

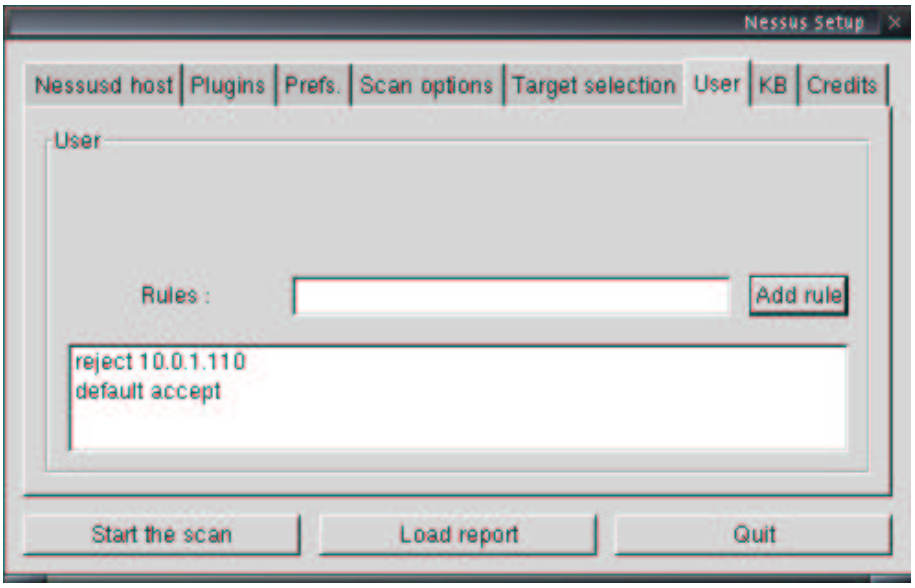
1.2.2. Creación de usuarios

Como ya hemos comentado anteriormente, para poder contactar con el servidor de Nessus desde el cliente, es necesario la utilización de usuarios. Mediante la utilización de la aplicación `nessus-adduser` será posible la definición de nuevos usarios desde sistemas GNU/Linux. Esta utilidad nos permitirá crear tanto usuarios locales, que tan sólo podrán acceder a Nessus desde el equipo local, como usuarios remotos, que podrán acceder a Nessus desde máquinas remotas.

Por otro lado, Nessus también ofrece la posibilidad de configurar distintos parámetros asociados con dichos usuarios: listado de máquinas desde las que se podrán conectar los usarios remotos, listado de *plug-ins* que se les permitirá ejecutar, listado de máquinas que el usuario podrá analizar, etc. La posibilidad de poder modificar estas conductas de exploración responden a la posibilidad de tener diferentes perfiles de administradores, responsables de la realización de distintos tipos de exploración según sus privilegios en el sistema.

Aunque para la creación tanto de usuarios locales como de usuarios remotos será posible la utilización del comando `nessus-adduser`, la configuración de las conductas de exploración u otras características más específicas deberán ser especificadas de distintas maneraras. Algunas, como la lista de máquinas desde las que se podrá conectar el usuario, deberán ser especificadas de forma manual en los ficheros de configuración correspondientes a cada usuario (generalmente, en `/etc/nessus/`).

A través del cliente de Nessus también será posible realizar un control personal para su utilización como, por ejemplo, limitar el número de máquinas que podrán ser analizadas por el usuario. La siguiente figura muestra cómo limitar el cliente para que pueda hacer una exploración a cualquier máquina excepto al equipo `10.0.1.110`.



Resumen

En este capítulo hemos visto cómo utilizar dos poderosas herramientas basadas en código abierto para la realización de exploración de puertos y exploración de vulnerabilidades en red. Mediante la primera de estas dos herramientas, Nmap, es posible descubrir qué puertos TCP o UDP están ofreciendo servicios en equipos explorados. Pero Nmap no es tan sólo una herramienta para descubrir servicios abiertos, Nmap ofrece muchas otras características como, por ejemplo, descubrir el sistema operativo albergado por dichos equipos, las características de implementación de la pila TCP/IP de tales sistemas operativos, la posibilidad de realizar predicción de números de secuencia TCP, realizar comprobaciones contra *routers* o *firewalls* intermedios, etc. Nmap es una herramienta muy importante a conocer, ya que es utilizada por la mayor parte de la comunidad de administradores de *software* libre. Además, es utilizada por terceras aplicaciones como, por ejemplo, Nessus.

La segunda herramienta estudiada, Nessus, es un potente escáner de vulnerabilidades basado en código libre, que proporcionará al administrador la posibilidad de realizar complejos análisis de red para detectar vulnerabilidades de seguridad en equipos remotos. Basado en una arquitectura cliente-servidor, Nessus ofrecerá la posibilidad de realizar exploraciones proactivas en busca de servidores antiguos o mal configurados, deficiencias de seguridad en la implementación TCP/IP de equipos en producción, instalación de servicios ilegítimos o sospechosos en los equipos de nuestra red, etc.

Por un lado, el cliente de Nessus hará de interfaz intermedia entre el administrador de la red y la aplicación que realizará los distintos chequeos de seguridad (servidor de Nessus). De este modo, podremos ejecutar las exploraciones de vulnerabilidades desde el exterior de la red, utilizando distintos sistemas operativos donde recoger los resultados y construir los informes con la información reportada por el servidor de Nessus. Por otro lado, será posible la instalación de diferentes instancias del servidor de Nessus en distintas localizaciones de la red, para realizar las exploraciones de seguridad con distintas vistas al sistema.

Nessus ofrece también un conjunto de posibilidades de autenticación de usuarios, para garantizar que usuarios no autorizados no utilicen los recursos de la red para realizar exploraciones de puertos o de vulnerabilidades de forma ilegítima. A través del uso de contraseñas de usuario, o bien mediante el uso de técnicas criptográficas, el cliente Nessus realizará un proceso de autenticación contrá el servidor de Nessus para garantizar que el usuario que se conecta a dicho servicio es un usuario legítimo.

Finalmente, Nessus ofrece la posibilidad de almacenar los resultados de la exploración realizada. Un amplio rango de formatos es ofrecido por el cliente de Nessus para almacenar los informes entregados por el sevidor de Nessus. Desde un formato propio de aplicación, hasta la utilización de formatos como ASCII, HTML, XML, LaTeX, etc.

Bibliografía

[1] Eyler, P. (2001). *Networking Linux: A Practical Guide to TCP/IP.* New Riders Publishing.

[2] Stanger, J.; Lane, P. T.; Danielyan E. (2001). *Hack Proofing Linux: A Guide to Open Source Security.* Syngress Publishing, Inc.

[3] Toxen, B. (2000). *Real World Linux Security: Intrusion Prevention, Detection, and Recovery.* Prentice Hall PTR.

A2 – Filtrado y registro de paquetes con Iptables

Joaquín García Alfaro

Índice

2.1. Iptables

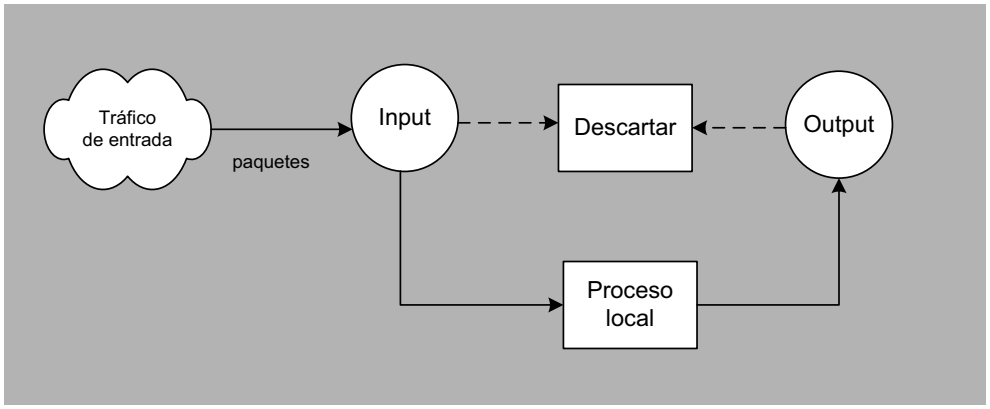
La herramienta iptables permite el filtrado y monitorización de tráfico TCP/IP en sistemas GNU/Linux. De hecho, esta herramienta no es más que una interfaz hacia el módulo netfilter de la serie 2.4 o superior del kernel Linux, el cual proporciona los mecanismos de seguridad para la capa de red del kernel (como, por ejemplo, el filtrado de paquetes), así como otras funciones relacionadas con el tratamiento de paquetes TCP/IP como, por ejemplo, traduccion de direcciones de red (Network Address Translation, NAT).

Origen de iptables

iptables fue escrito por Rusty Russell, también autor de ipchains (módulo de la serie 2.2 del kernel Linux para el tratamiento de paquetes TCP/IP). Su trabajo fue patrocinado por Watchguard (www.watchguard.com), Penguin Computing (antarctica.penguincomputing.com/ netfilter/), el equipo “Samba Team” (www.samba.org/netfilter/) y Jim Pick (netfilter.kernelnotes.org). El equipo “Samba Team” también mantiene una lista de correo sobre iptables. Ver lists.samba.org para más información.

Como ya hemos avanzado, netfilter es un modulo para la serie 2.4 del kernel Linux y es el responsable del filtrado de paquetes TCP/IP. Observando las cabeceras de cada paquete que pasa por el equipo, decidirá si ese paquete debe ser aceptado, descartado o si deberá realizarse alguna otra operación más compleja con él.

Para realizar filtrado de paquetes en un único equipo, Netfilter parte de dos conjuntos de reglas básicas: INPUT y OUTPUT. De ahora en adelante nos referiremos a estos conjuntos como cadenas de filtrado o simplemente cadenas. Los paquetes TCP/IP pasarán a través de estas cadenas básicas tal y como muestra la siguiente figura:



Cada cadena tendrá asociada una lista de reglas que serán consultadas de forma secuencial por cada paquete. A medida que éstos vayan atravesando las reglas de la cadena que les corresponda, serán examinados para determinar qué acción tomar. En caso de ser aceptados, pasarán al siguiente punto del diagrama. Por contra, si son descartados, el paquete desaparecerá del esquema.

2.1.1. Utilización de la herramienta iptables

El primer paso para dar de alta iptables es asegurarse de que ha sido creado correctamente como una parte del kernel o como módulos asociados al mismo. Para ello, habrá que contestar afirmativamente a la cuestión CONFIG_NETFILTER durante la configuración de un kernel 2.3.15 o superior.

Si iptables ha sido construido correctamente, podremos utilizar las herramientas asociadas en el espacio de usuario. La herramienta básica es iptables, que será utilizada para la construcción del conjunto de reglas. Estos conjuntos de reglas se añadirán a cada una de las cadenas básicas como si fueran bloques independientes de filtrado. En el siguiente ejemplo podemos ver un guión de sistema para la construcción de un conjunto de reglas de filtrado y registro de paquetes mediante iptables:

```
# creamos una nueva cadena
iptables -N EJEMPLO


# activamos algunas reglas en la cadena EJEMPLO
iptables -A EJEMPLO -i eth0 -s 10.0.1.100 -j LOG --log-prefix
"Paquete descartado:"
iptables -A EJEMPLO -s 10.0.1.100 -j DROP
iptables -A EJEMPLO -s 10.0.2.100 -j LOG --log-prefix
"Paquete aceptado:"
iptables -A EJEMPLO -j LOG --log-prefix "Aceptando paquete:"
iptables -A EJEMPLO -j DROP

# todo el tráfico que nos llegue a la cadena de INPUT saltará
# a la cadena EJEMPLO
iptables -A INPUT -j EJEMPLO
```

Después de cargar el conjunto de reglas que mostramos en el ejemplo anterior, podríamos tratar de realizar una conexión desde las direcciones IP 10.0.1.100 y 10.0.2.100. La primera conexión fallará (y generará una entrada de registro en el equipo de destino indicándolo), mientras que la segunda conexión será aceptada (generando también una nueva entrada de registro). Para desactivar las reglas y dejar el equipo tal y como estaba antes de lanzar el guión de sistema, realizaremos las siguientes llamadas a iptables:

```
iptables -F EJEMPLO
iptables -X EJEMPLO
iptables -F INPUT
```

Una vez nos hemos asegurado de que el kernel contiene el código de iptables correctamente, y hemos verificado la existencia de la herramienta necesaria para acceder a él, veremos a continuación cómo construir reglas de filtrado mediante iptables.

2.1.1.1. Comandos para la manipulación de cadenas

Antes de empezar a escribir reglas de iptables, es importante tener claro la política de seguridad que queremos implementar, cómo construir las reglas de manera que sean fáciles de manejar y de mantener, y cómo hacer que las reglas sean lo más eficientes posible sin perder por ello las facilidades anteriores.

Si las reglas que vamos a dar de alta no implementan de forma correcta nuestra política de seguridad, más valdría no activarlas. Por otro lado, es muy probable que si las reglas no han sido construidas de forma correcta, un atacante detecte cómo vulnerarlas para alcanzar su objetivo. Además, la construcción de reglas complejas y de difícil comprensión aumentará la posibilidad de cometer entradas erróneas que dejen abierta una brecha en el sistema.

Antes de empezar a trabajar con iptables, es importante repasar los distintos parámetros de la herramienta. La siguiente tabla muestra un resumen de los comandos básicos de iptables para el tratamiento de cadenas:

Parámetro	Descripción
-N cadena	Creación de una nueva cadena
-L cadena	Muestra las reglas de una cadena
-P cadena	Cambia la política por defecto de una cadena
-Z cadena	Inicializa el contador de paquetes de una cadena
-F cadena	Vacia todas las reglas de una cadena
-X cadena	Elimina una cadena si está vacia (excepto las cadenas básicas)

Aparte de las operaciones básicas para el mantenimiento y creación de cadenas, también necesitaremos un conjunto de operaciones para el tratamiento de reglas de una cadena. La siguiente tabla muestra algunas de las operaciones básicas de iptables para este propósito:

Parámetro	Descripción
-A cadena regla	Añade una regla al final de la cadena
-I cadena regla pos.	Inserta una regla en la posicion indicada de la cadena
-R cadena regla pos.	Reemplaza una regla en la posicion indicada de la cadena
-D cadena regla pos.	Elimina una regla de la cadena (puede indicarse por posicion o por contenido)

Utilizando las opciones de cadenas y de reglas que hemos visto en las tablas anteriores, podremos empezar a crear y modificar el guión de sistema que habíamos iniciado en la sección anterior. Veamos, por ejemplo, como deberíamos analizar la siguiente entrada:

```
iptables -A EJEMPLO -s 10.0.1.100 -j DROP
```

El parámetro `-A EJEMPLO` indica que estamos añadiendo una nueva regla a la cadena `EJEMPLO` y que dicha regla será añadida al final de la cadena. La opción `-s 10.0.1.100` indica que la regla será aplicada únicamente a aquellos paquetes cuya dirección IP de origen sea `10.0.1.100`.

Por último, el parámetro `-j DROP` le indicará a `netfilter` que descarte este paquete sin ningún procesado posterior. Es decir, el parámetro `-j` indica a netfilter que el paquete que acaba de entrar por la cadena `EJEMPLO` debe saltar a la cadena `DROP`. Todos los paquetes que lleguen a la cadena `DROP`, serán descartados.

2.1.1.2. Filtrado por dirección IP

En nuestro ejemplo, hemos definido la dirección de origen mediante el parámetro `-s`. Esta opción de iptables puede ser definida mediante el uso de máscaras como, por ejemplo:

```
iptables -A EJEMPLO -s 10.0.1.0/24 -j DROP
```

o mediante el nombre de equipo:

```
iptables -A EJEMPLO -s equipo1.ejemplo.es -j DROP
```

De manera similar a la dirección de origen, podríamos realizar el filtrado mediante la dirección de destino utilizando como argumento el parámetro `-d` (los argumentos `-source`, `-src`, `-destination` y `-dst` también serán válidos).

2.1.1.3. Filtrado por interfaz de red

En lugar de filtrar por dirección IP, podríamos definir filtros de paquetes por interfaz de red. Las opciones de iptables `-i` y `-in-interface` definen interfaces de entrada, mientras que las opciones `-o` y `-out-interface` definen interfaces de salida.

Por otro lado, aparte de las cadenas básicas ya comentadas anteriormente, existen otras dos cadenas predefinidas, la cadena `INBOUND` y la cadena `OUTBOUND`, que estarán asociadas por defecto a los paquetes que pasen por las interfaces de entrada y de salida del equipo. También es posible el uso de comodines para indicar el nombre de las interfaces como, por ejemplo, el comodín `+` del siguiente ejemplo:

```
-o eth+
```

Mediante el comodín `+` podemos indicar cualquier interfaz asociada a la cadena que le precede. En el ejemplo anterior, cualquier paquete que salga al exterior por una interfaz de red cuyo nombre empiece por la cadena `eth` (cualquier interfaz de red Ethernet) será tratado por la regla, asociada.

2.1.1.4. Filtrado por protocolo

Para realizar el filtrado a través de protocolo podemos utilizar la opción `-p` de iptables. Para especificar el protocolo, podremos hacerlo bien por el identificador del protocolo (tal y como se especifique en el RFC de la familia de protocolos TCP/IP) o bien por su nombre asociado (como, por ejemplo, TCP, UDP e ICMP). Para especificar, por ejemplo, cualquier paquete de ICMP podríamos utilizar la siguiente opción de iptables:

```
-p ICMP
```

2.1.2. Invirtiendo selecciones

En muchas situaciones es más sencillo indicar algo por eliminación que tener que indicarlo de manera individual. Por ejemplo, podría ser más sencillo indicar que todos los paquetes excepto los que provengan de la red IP 10.0.1.100/24 sean asociados a una regla, que tener que indicar de forma explícita todos aquellos equipos que no lo son. Para ello, iptables utilizará el prefijo de inversión `!` para permitirnos la inversión de una condición. Así, para seleccionar los paquetes que antes indicábamos, una entrada de iptables es la siguiente:

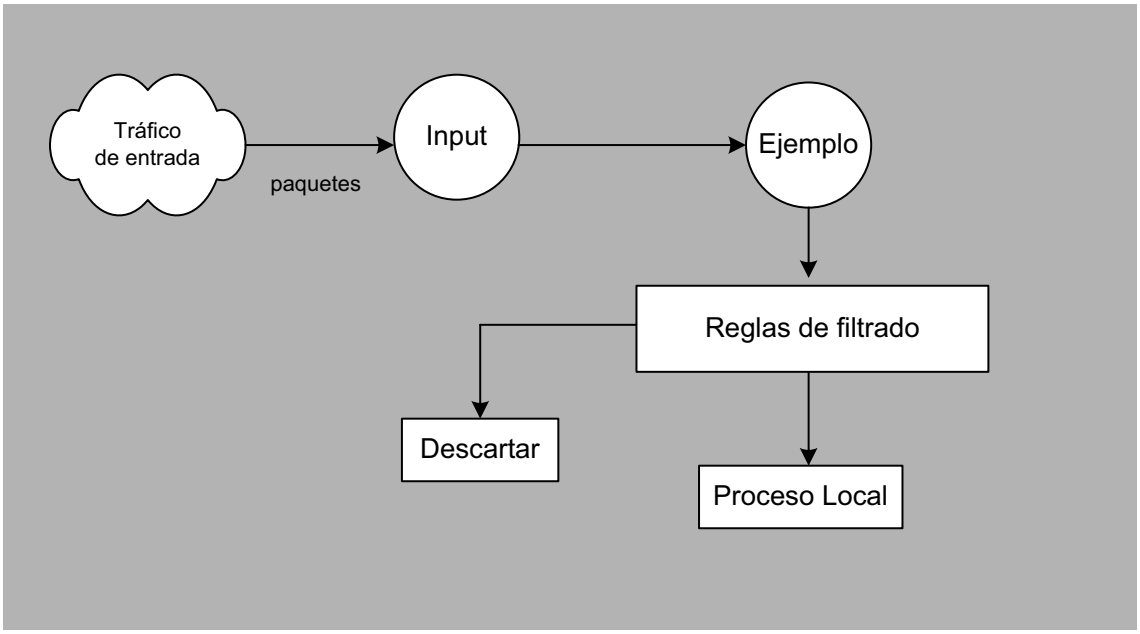
```
-s !  10.0.1.0/24
```

2.1.3. Ejemplo de definición y utilización de cadenas

Volviendo de nuevo a nuestro guión de sistema de ejemplo para la construcción de reglas de filtrado,

```
# creamos una nueva cadena
iptables -N EJEMPLO


# activamos algunas reglas en la cadena EJEMPLO
iptables -A EJEMPLO -i eth0 -s 10.0.1.100 -j LOG --log-prefix
"Paquete descartado:"
iptables -A EJEMPLO -s 10.0.1.100 -j DROP
iptables -A EJEMPLO -s 10.0.2.100 -j LOG --log-prefix
"Paquete aceptado:"
iptables -A EJEMPLO -j LOG --log-prefix "Aceptando paquete:"
iptables -A EJEMPLO -j DROP

# todo el tráfico que nos llegue a la cadena de INPUT saltará
# a la cadena EJEMPLO
iptables -A INPUT -j EJEMPLO
```

vemos la creación de una cadena de ejemplo sobre la que asociamos los paquetes que pasen por la cadena INPUT del equipo. Así, el tráfico de paquetes que pase por la cadena INPUT fluirá de la siguiente manera



Observando de nuevo el guión de sistema, podemos identificar los siguientes pasos:

1) Creación de la cadena `EJEMPLO` mediante la opción `iptables -N EJEMPLO`. Aunque el nombre de la cadena no debe ser forzosamente en letras mayúsculas, sí es un requerimiento que no exista ninguna otra cadena de igual nombre.

2) Se añaden una serie de reglas asociadas a la cadena `EJEMPLO` de forma secuencial, aunque también lo podríamos haber hecho imponiendo un orden a cada regla mediante la sentencia `-I`.

3) Una vez que se han escrito las reglas para la cadena `EJEMPLO`, añadimos una nueva regla para asociar el tráfico que pasa por la cadena `INPUT` hacia la cadena `EJEMPLO` mediante la sentencia `iptables -A INPUT -j EJEMPLO`. De este modo, todo el tráfico de la cadena `INPUT` saltará a la cadena `EJEMPLO` para que sea procesado posteriormente por las reglas de esta cadena.

Mediante el comando `iptables -L EJEMPLO` veríamos el siguiente resultado:

```
Chain EJEMPLO (1 references)
target prot opt source destination LOG all -- 10.0.1.100 0.0.0.0/0
LOG flags 0 level 4 prefix ‘Paquete descartado:’
DROP all -- 10.0.1.100 0.0.0.0/0
LOG all -- 10.0.2.100 0.0.0.0/0
LOG flags 0 level 4 prefix ‘Paquete aceptado:’
LOG all -- 0.0.0.0/0 0.0.0.0/0
LOG flags 0 level 4 prefix ‘Aceptando paquete:’
DROP all -- 0.0.0.0/0 0.0.0.0/0
```

2.1.4. Tratamiento adicional

En algunas ocasiones puede ser interesante poder registrar el tráfico que circula por nuestro filtro de paquetes. Pero este tipo de registro podría llegar a consumir una gran cantidad de espacio en disco, por lo que a menudo suele realizarse del modo más reducido posible. En nuestro ejemplo, conseguimos este objetivo mediante la siguiente sentencia:

```
iptables -A EJEMPLO -j LOG --log-prefix "Aceptando paquete:"
```

De este modo, todos los paquetes que fluyan por la cadena `EJEMPLO` serán registrados en un fichero de registro con la cadena de texto `Aceptando paquete:` precediendo a la información de registro proporcionada por iptables. Otra opción muy utilizada para el registro de paquetes mediante reglas de `iptables` suele la opción `--log-level`, con la cual podremos asociar enviar el registro de paquetes en forma de evento de `syslog`.

Otra característica a tener presente es que cuando un paquete es descartado, esta operación se hace de modo silencioso (sin avisar al origen que el paquete ha sido descartado). Si,

por el contrario, nos interesa que un mensaje de tipo `ICMP Port Unreachable` fuese enviado al origen indicando que el paquete no ha sido aceptado, podemos utilizar la opción `-j REJECT` para conseguir dicho propósito.

Por último, si queremos utilizar conjuntamente múltiples cadenas, es posible no finalizar una cadena indicando que descarte los paquetes asociados a ella. En su lugar, es posible pasar el control del paquete hacia la cadena de origen (en nuestro ejemplo, hacia la cadena INPUT) mediante la opción `-j RETURN`. Si la cadena estuviese a nivel superior y no tuviera, por tanto, ninguna predecesora, lo que hará netfilter es ejecutar la política por defecto de la cadena.

Firestarter

A la hora de utilizar Firestarter hay que tener presente que se trata de un *front-end* basado en código libre para la creación de guiones de filtrado. De este modo, es posible revisar el código de Firestarter en busca de posibles errores o *bugs* en la aplicación antes de su utilización en un entorno de producción. Al igual que otras interfaces gráficas para Iptables (o Ipchains), este tipo de aplicaciones suelen estar disponibles en versiones beta, por lo que es muy posible que su funcionalidad esté todavía limitada.

2.1.5. Uso y obtención de guiones de filtrado mediante herramientas gráficas

Existen un gran número de interfaces gráficas para la creación automática de guiones de sistema de cara a efectuar la realización de filtrado de una forma más cómoda e intuitiva. Aunque la mayoría de estas aplicaciones son realmente de gran utilidad, en muchas ocasiones será necesario tener que ajustar los guiones que dichas herramientas generan de forma automática. A continuación veremos la utilización de una de estas herramientas. En concreto, veremos la construcción de un guión de filtrado generado automáticamente mediante la herramienta Firestarter.

Firestarter es una interfaz gráfica para la construcción automática de guiones de filtrado tanto para Iptables como para Ipchains. Es una herramienta de gran utilidad para la creación de un sistema cortafuegos sencillo, aunque también soporta la creación de guiones de filtrado más complejos (con más de una interfaz de red en la mayoría de las ocasiones).

Antes de empezar a utilizar Firestarter, en caso de no disponer de la aplicación ya instalada en el sistema, deberemos descargar el código de la aplicación desde la ubicación correspondiente y compilar el código fuente. Firestarter puede ser descargado libremente desde `http://firestarter.sourceforge.net/`. En la misma página, se pueden encontrar las instrucciones correspondientes para la instalación de la aplicación en entornos GNU/Linux y desde diferentes distribuciones (en forma de paquetes RPM, de paquetes DEB, etc.).

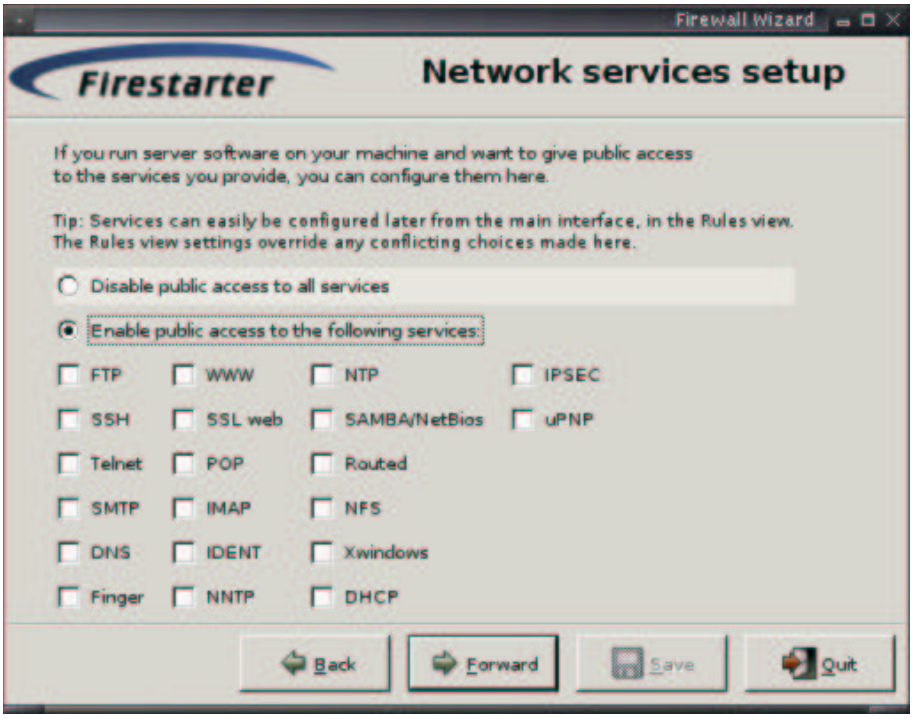
Una vez instalado en el sistema Firestarter, ejecutaremos la aplicación desde una terminal con privilegios de usuario *root* mediante el comando `firestarter`, o desde la entrada correspondiente en el menú de aplicaciones del sistema. En caso de ejecutar la aplicación con privilegios de usuario distintos a los de *root*, la aplicación solicitará la contraseña de *root* de forma interactiva para poder continuar adelante.

Si la aplicación se ejecuta por primera vez, nos aparecerá en pantalla el asistente de la aplicación. Como podemos ver en la siguiente figura, el asistente de Firestarter nos guiará para realizar una configuración inicial del conjunto de guiones de filtrado. Más adelante, el guión podrá ser modificado para ajustarse mejor a las necesidades del sistema.

La siguiente pantalla de la aplicación solicitará las interfaces de red a configurar. La aplicación detectará las interfaces de red activas y esperará que le indiquemos cuál es la interfaz externa, es decir, la interfaz que conecta nuestro equipo con Internet. En caso de disponer de una única interfaz de red, no habrá ningún problema con seleccionar la interfaz mostrada por defecto. Por otro lado, en caso de estar conectado a Internet utilizando direcciones IP asignadas dinámicamente, deberemos seleccionar la opción DHCP para que Firestarter lo tenga presente en los guiones que generará.

En caso de realizar la conexión vía módem, es posible que desde Firestarter deba indicarse `ppp0` como interfaz de conexión a Internet. Por otro lado, en caso de realizar la conexión a Internet a través de ADSL, es también posible que debamos indicar a Firestarter que `ppp0` es la interfaz de conexión al exterior, pues muchos ISP utilizan el protocolo `PPPoE` para la conexión a Internet. Así pues, si desde el asistente de Firestarter se nos muestra la interfaz `ppp0` y utilizamos módem o una conexión ADSL a través de protocolo `ppp0`, deberemos seleccionar la interfaz `ppp` de la lista ofrecida.

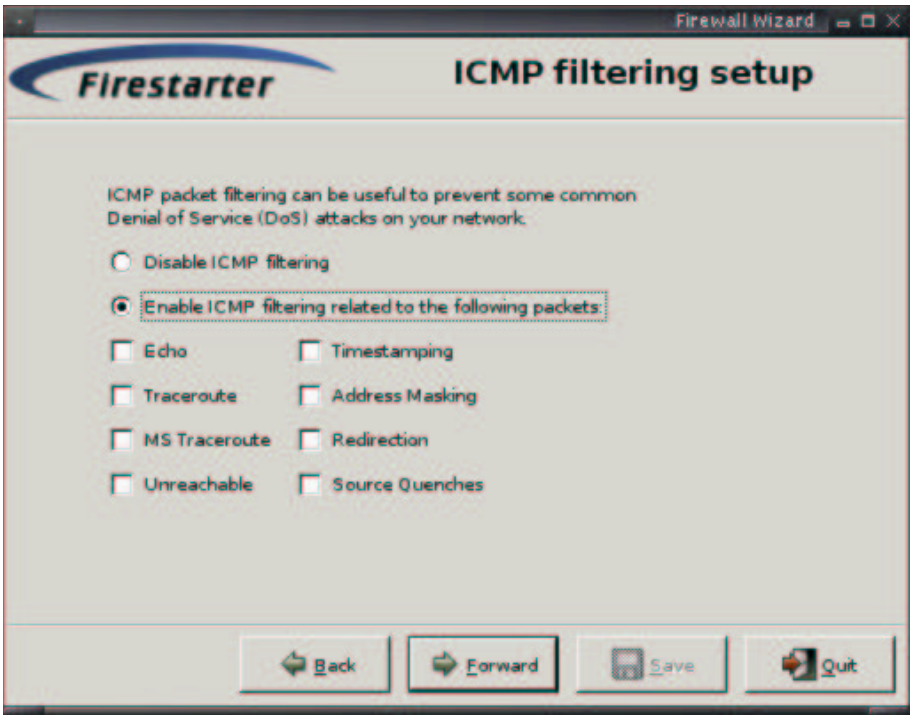
A continuación, el asistente de Firestarter nos mostrará una pantalla donde indicar cuáles de los servicios que nuestro equipo está ofreciendo serán accesibles desde el exterior. De este modo, si nuestro equipo está ejecutando servidores como, por ejemplo, un servidor de HTTP o un servidor de DNS, podremos indicar desde esta pantalla si tales servicios permanecerán accesibles desde Internet, o por contra, si el acceso a dichos servicios será filtrado por Iptables o por Ipchains. Más adelante, una vez finalizado el asistente de Firestarter, podremos ajustar de manera más exhaustiva qué equipos del exterior podrán acceder a tales servicios.



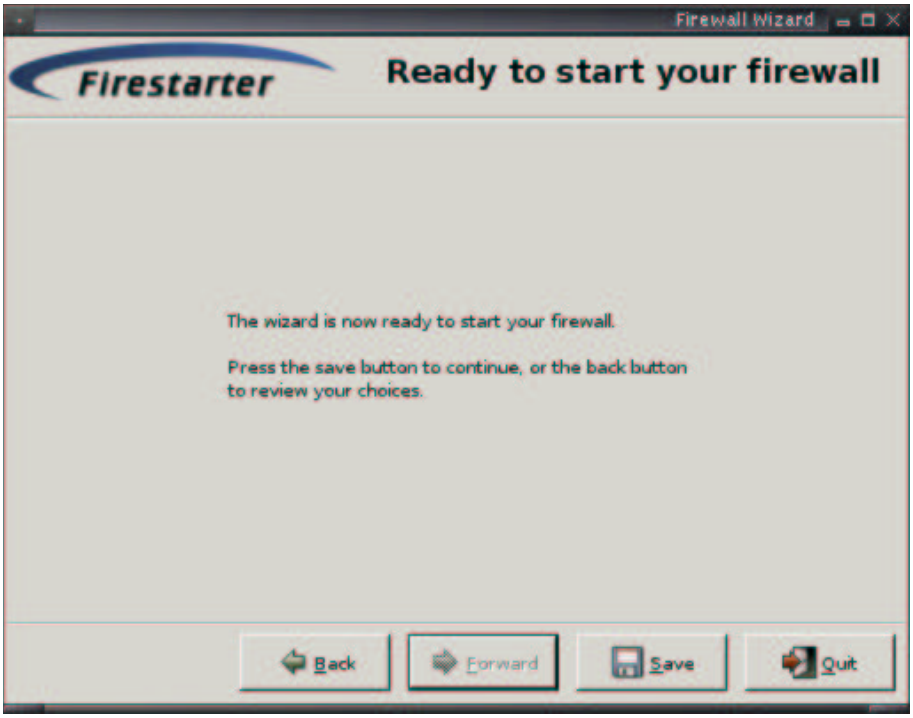
La siguiente pantalla muestra cómo realizar un filtrado dedicado al tráfico ICMP que nuestro equipo aceptará desde el exterior. Por defecto, Firestarter permitirá cualquier tipo de tráfico ICMP. Hay que tener presente que filtrar tráfico ICMP puede comportar que algunas aplicaciones de administración como, por ejemplo, *ping* o *traceroute*, puedan dejar de funcionar correctamente al tratar de alcanzar a nuestro equipo desde el exterior. Así, para evitar que usuarios desde el exterior de nuestra red puedan utilizar estas aplicaciones contra nuestro equipo, será conveniente indicar al asistente de Firestarter que realice tal

filtrado.

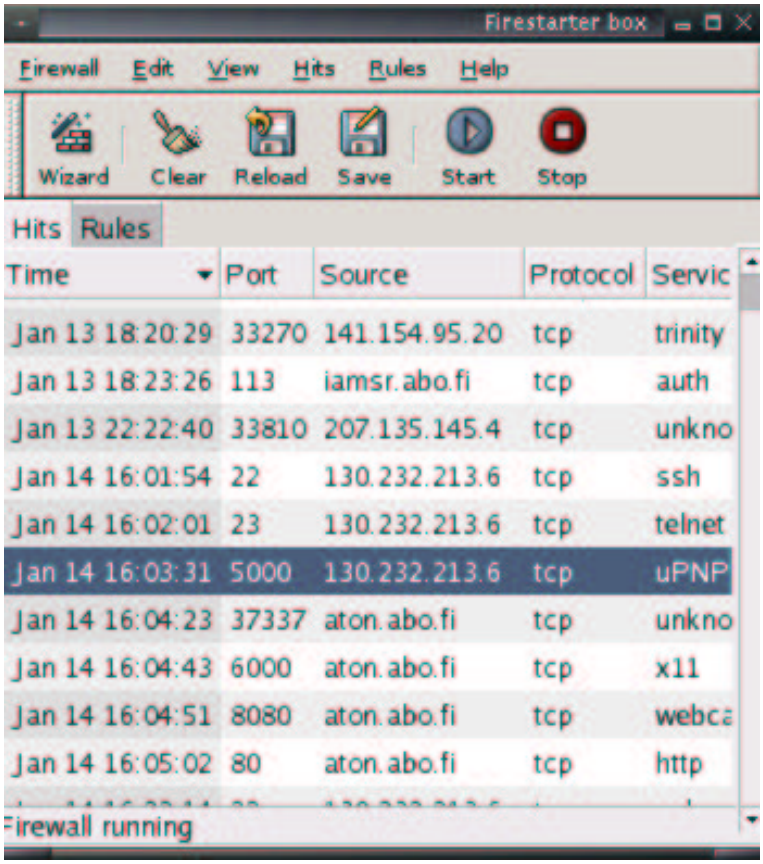
Como podemos ver en la siguiente figura, Firestarter ofrecerá la posibilidad de bloquear los diferentes tipos de mensajes utilizados por ICMP: *echo*, *traceroute*, *MS traceroute*, *unreachable*, *timestamping*, *address masking*, *redirection* y *source quenches*.



Finalmente, el asistente de Firestarter mostrará una pantalla indicando la creación de los guiones de filtrado correspondientes a las contestaciones que hayamos ido entrando a lo largo de las pantallas anteriores.



A continuación, será conveniente visitar las pantallas correspondientes a las pestañas *Hits* y *Rules* de la aplicación. Desde la primera de estas dos pantallas, *Hits view*, podremos visualizar el tráfico que está siendo bloqueado por nuestro equipo una vez que la aplicación ha activado las reglas de filtrado que han sido generadas por el asistente de Firestarter.



La segunda pantalla, *Rules view*, nos permitirá ajustar las reglas creadas previamente por el asistente, así como eliminar o añadir nuevas entradas. Estas reglas pertenecen tan sólo a la aplicación Firestarter, y no debe confundirse con las reglas asociadas a las cadenas de netfilter. Es una forma particular que tiene la aplicación para agrupar cierta información que más adelante utilizará para la construcción de reglas de netfilter. Este conjunto de reglas están divididas en cinco grupos:

- *Trusted hosts* - los equipos listados en esta categoría serán considerados de confianza, es decir, que tendrá libre acceso a todos los servicios sin restricciones por parte de las reglas de filtrado.

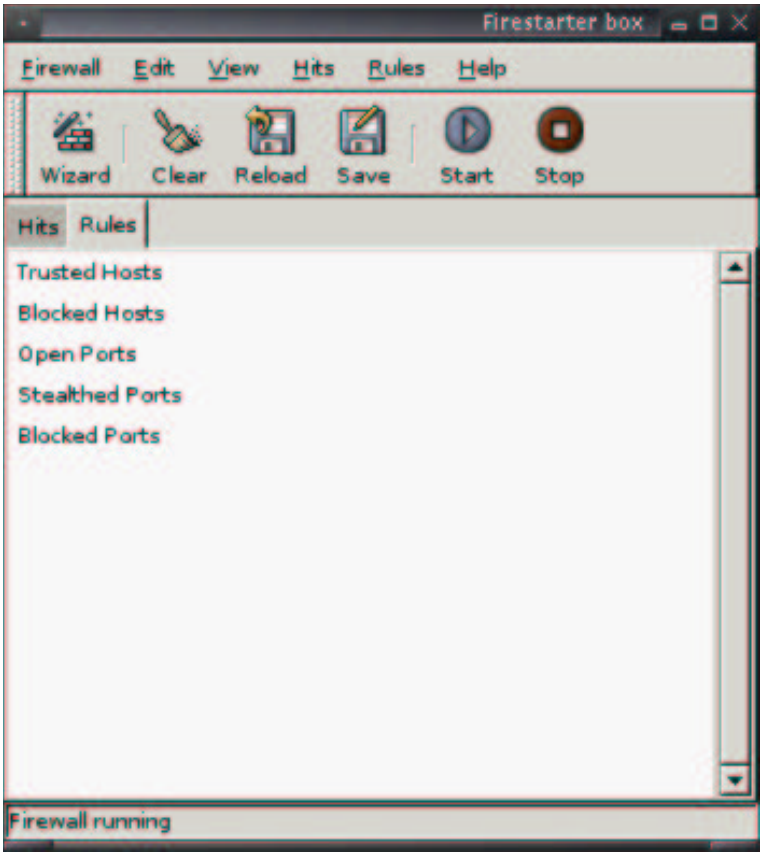
- *Blocked hosts* - equipos que serán considerados peligrosos y cuyo tráfico (cualquiera) será bloqueado (rechazado) por los guiones de filtrado de Firestarter. Tan pronto como se añada un equipo dentro de esta categoría, cualquier intento de conexión por su parte será bloqueado. Las conexiones de tales equipos no serán mostradas en la pantalla de *Hits*.

- *Open ports* - Esta categoría muestra servicios (puertos) que serán libremente accesibles por cualquier equipo del exterior (excepto por los equipos incluidos en la categoría anterior). Por ejemplo, una entrada en la categoría de `open ports` correspondiente a HTTP (por defecto, el puerto 80 de TCP) indicará que cualquier equipo del exterior, excepto los indicados en *bloqued hosts*, podrá acceder a nuestro servidor de HTTP.

- *Stealthed ports* - esta categoría contendrá puertos que aparecerán ocultos para cualquier equipo, excepto para los equipos indicados dentro de las reglas de esta categoría. De alguna manera, esta categoría es similar a la categoría de equipos de confianza, pero indicando únicamente que los equipos incluidos en estas reglas son de confianza exclusivamente para el puerto indicado.

- *Blocked ports* - Por defecto, cualquier puerto que no aparezca explícitamente abierto en la categoría *open ports* será bloqueado y registrado por los guiones de filtrado de Firestarter. Esta categoría sirve para detener explícitamente intentos de conexión hacia un puerto sin que la acción sea registrada. Por ejemplo, si nos encontramos en una red saturada con gran cantidad de intentos de conexión hacia el servicio Netbios de equipos Windows, podríamos incluir los puertos 137 y 138 en esta categoría, para filtrar el acceso a dichos puertos sin necesidad de generar una entrada de registro por cada intento.

La siguiente figura muestra la interfaz para la pantalla de reglas. Para añadir una regla en cualquiera de las categorías, sólo será necesario apuntar sobre la categoría deseada y entrar los parámetros necesarios para la construcción de la regla. Los cambios serán efectuados tan pronto como la regla sea añadida en dicha pantalla.



Finalmente, si ejecutamos el comando `iptables -L` con los nombres de cadena correspondientes, podremos observar las reglas de filtrado que Firestarter habrá creado. La siguiente figura muestra el contenido de las cadenas `OUTPUT` y `LD` (tal y como Firestarter ha creado).

```
#iptables -L OUTPUT -n

Chain OUTPUT (policy DROP)
target     prot opt source               destination
UNCLEAN    all  --  0.0.0.0/0            0.0.0.0/0          unclean
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:31337 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:31337 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:33270 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:33270 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:1234 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:6711 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:16660 flags:0x16/0x02 limit: avg 2/min
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:60001 flags:0x16/0x02 limit: avg 2/min
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpts:12345:12346 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpts:12345:12346 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:135 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:135 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:1524 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:27665 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:27444 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:31335 limit: avg 2/min burst 5
LD         all  --  255.255.255.255      0.0.0.0/0
LD         all  --  0.0.0.0/0            0.0.0.0
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0          tcp flags:!0x16/0x02 state NEW
           all  --  0.0.0.0/0            0.0.0.0/0          TTL match TTL == 64
ACCEPT     icmp --  10.0.1.0/24      0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

#iptables -L LD -n
Chain LD (146 references)
target     prot opt source               destination
LOG        all  --  0.0.0.0/0            0.0.0.0/0          LOG flags 0 level 4
DROP       all  --  0.0.0.0/0            0.0.0.0/0
```

Hay que tener presente que cualquier regla entrada manualmente por el administrador del sistema deberá ser incluida en la configuración realizada por Firestarter, o se perderá al

reiniciar el sistema. Respecto al guión de filtrado generado por Firestarter y, por tanto, responsable de lanzar las reglas de filtrado mostradas anteriormente, éste se encuentra generalmente en el fichero `/etc/firestarter/firewall.sh`. Las dos figuras siguientes muestran el inicio de dicho fichero.

```
#ls -lah /etc/firestarter/
total 44K
drwx------    2 root     root         4.0K May 11 09:15 .
drwxr-xr-x  163 root     root         8.0K May 11 09:40 ..
-rwx------    1 root     root            0 Feb  2 00:48 blocked-hosts
-rwx------    1 root     root            0 Feb  2 00:48 blocked-ports
-rwx------    1 root     root          25K Feb  2 00:48 firewall.sh
-rwx------    1 root     root            0 Feb  2 00:48 forward
-rwx------    1 root     root            0 Feb  2 00:48 open-ports
-rwx------    1 root     root            0 Feb  2 00:48 stealthed-ports
-rwx------    1 root     root           11 Feb  2 00:49 trusted-hosts

#cat /etc/firestarter/firewall.sh

#!/bin/sh
# Generated by Firestarter 0.9.2, NETFILTER in use

# --------( Initial Setup - Variables (required) )--------

# Type of Service (TOS) parameters
# 8: Maximum Throughput - Minimum Delay
# 4: Minimize Delay - Maximize Reliability
# 16: No Delay - Moderate Throughput - High Reliability

TOSOPT=8

# Default Packet Rejection Type
# ( do NOT change this here - set it in the GUI instead )

STOP=DENY
```

```
# --------( Initial Setup - Firewall Location Check )--------
IPT=/sbin/iptables
IFC=/sbin/ifconfig
MPB=/sbin/modprobe
LSM=/sbin/lsmod
RMM=/sbin/rmmod

# --------( Initial Setup - Network Information (required) )--------
IF=eth0
IP=`$IFC $IF | grep inet | cut -d : -f 2 | cut -d \  -f 1`
MASK=`$IFC $IF | grep Mas | cut -d : -f 4`
NET=$IP/$MASK

if [ "$MASK" = "" ]; then
       echo "External network device $IF is not ready. Aborting.."
       exit 2
fi

# --------( Initial Setup - Firewall Modules Check )--------
# Some distributions still load ipchains
$LSM | grep ipchains -q -s && $RMM ipchains

# --------( Initial Setup - Firewall Modules Autoloader )--------
if ! ( $LSM | /bin/grep ip_conntrack > /dev/null ); then
$MPB ip_conntrack
fi
if ! ( $LSM | /bin/grep ip_conntrack_ftp > /dev/null ); then
$MPB ip_conntrack_ftp
fi
if ! ( $LSM | /bin/grep ip_conntrack_irc > /dev/null ); then
$MPB ip_conntrack_irc
fi
if ! ( $LSM | /bin/grep ipt_REJECT > /dev/null ); then
$MPB ipt_REJECT
fi
if ! ( $LSM | /bin/grep ipt_REDIRECT > /dev/null ); then
$MPB ipt_REDIRECT
fi
if ! ( $LSM | /bin/grep ipt_TOS > /dev/null ); then
$MPB ipt_TOS
fi
if ! ( $LSM | /bin/grep ipt_MASQUERADE > /dev/null ); then
$MPB ipt_MASQUERADE
fi
if ! ( $LSM | /bin/grep ipt_LOG > /dev/null ); then
$MPB ipt_LOG
fi
if ! ( $LSM | /bin/grep iptable_mangle > /dev/null ); then
$MPB iptable_mangle
fi
if ! ( $LSM | /bin/grep iptable_nat > /dev/null ); then
$MPB iptable_nat
fi
# --------( Chain Configuration - Flush Existing Chains )--------
# Delete user made chains. Flush and zero the chains.

$IPT -F
$IPT -X
$IPT -Z

# Remove Firestarter lock
if [ -e /var/lock/subsys ]; then
  rm -f /var/lock/subsys/firestarter
else
  rm -f /var/lock/firestarter
fi

···· ····· ····· ····

···· ····· ····· ····
```

Resumen

A lo largo de este capítulo hemos aprendido la utilización de una herramienta basada en *software* libre para la construcción de reglas de filtrado en sistemas GNU/Linux. Mediante la utilidad de administración `iptables`, será posible la comunicación con `netfilter`, es decir, el responsable de la manipulación de paquetes en un kernel Linux superior a la serie 2.3. `Netfilter` permite tanto el filtrado de paquetes, como la traducción de direcciones de red y otras manipulaciones más complejas.

Mediante `iptables` podremos crear un conjunto de reglas de filtrado para el tráfico de salida de nuestro equipo indicando, de forma explícita o implícita, qué tipo de paquetes de salida deberá rechazar o permitir el sistema. De igual manera, podremos crear un conjunto de reglas de filtrado de entrada, para indicar al sistema qué paquetes entrantes deberán ser aceptados o denegados.

Para la construcción de estas reglas de filtrado, `iptables` ofrece un amplio conjunto de opciones. Una buena manera de empezar a realizar estos guiones de filtrado consiste en la construcción de un guión que deniegue por defecto todo el tráfico de entrada y de salida, e ir indicando de forma explícita todos aquellos servicios a los que nuestro equipo podrá llegar, o todos aquellos servicios que nuestro equipo podrá ofrecer al exterior de la red. Otra de las distintas opciones de `iptables` que podemos utilizar es la funcionalidad de registro de paquetes. De este modo, podemos, por ejemplo, registrar en el sistema todo aquel tráfico de entrada o de salida que nuestro equipo vaya procesando.

Finalmente, hemos visto también en este capítulo la posibilidad de utilizar alguna herramienta gráfica para facilitar el proceso de creación de reglas de filtrado. Aunque existe un gran número de aplicaciones para la creación automática de reglas de filtrado, muchas de estas aplicaciones se encuentran aún en estado de desarrollo, por lo que es muy probable que los guiones de filtrado generados por tales aplicaciones deban ser ajustados de forma manual para asegurar su correcto funcionamiento. La herramienta *Firestarter*, por ejemplo, es una aplicación basada en *software* libre que nos ayudará a generar las reglas de filtrado de *netfilter/* de una forma mucho más sencilla.

Bibliografía

[1] Eyler, P. (2001). *Networking Linux: A Practical Guide to TCP/IP.* New Riders Publishing.

[2] Stanger, J.; Lane, P. T.; Danielyan E. (2001). *Hack Proofing Linux: A Guide to Open Source Security.* Syngress Publishing, Inc.

[3] Toxen, B. (2000). *Real World Linux Security: Intrusion Prevention, Detection, and Recovery.* Prentice Hall PTR.

A5 – Detección de ataques en red con Snort

Joaquín García Alfaro

Índice

5.1. Snort

Snort es una completa herramienta de seguridad basada en código abierto para la creación de sistemas de detección de intrusos en entornos de red. Cuenta con una gran popularidad entre la comunidad de administradores de redes y servicios. Gracias a su capacidad para la captura y registro de paquetes en redes TCP/IP, Snort puede ser utilizado para implementar desde un simple *sniffer* de paquetes para la monitorización del tráfico de una pequeña red, hasta un completo sistema de detección de intrusos en tiempo real.

Mediante un mecanismo adicional de alertas y generación de ficheros de registro, Snort ofrece un amplio abanico de posibilidades para la recepción de alertas en tiempo real acerca de los ataques y las intrusiones detectadas.

Tal y como indica el autor de la aplicación, como monitor de red, Snort se comporta como una auténtica aspiradora (de ahí su nombre) de datagramas IP, ofreciendo diferentes posibilidad en cuanto a su tratamiento. Desde actuar como un simple monitor de red pasivo que se encarga de detectar el tráfico maligno que circula por la red, hasta la posibilidad de enviar a servidores de ficheros de registro o servidores de base de datos todo el tráfico capturado.

Pero, aparte de unas estupendas características como *sniffer* de paquetes y generador de alertas e informes, Snort tiene muchas otras características que le han permitido convertirse en una de las soluciones *software* más completas para la construcción de sistemas de detección en entornos de red basados en reconocimiento de patrones. Snort debería considerarse como un NIDS ligero*. Este calificativo de *ligero* significa que, como IDS, su diseño e implementación le permite poder funcionar bajo diferentes sistemas operativos y que sus funciones como mecanismo de detección podrán formar parte en distintos productos de seguridad (incluso comerciales).

* En inglés, *lightweight NIDS (Network Intrusion Detection System)*.

La popularidad de Snort se ha incrementado estos últimos años en paralelo al incremento de popularidad de sistemas operativos de código abierto como puede ser el sistema operativo GNU/Linux o la familia de sistemas operativos de BSD (NetBSD, OpenBSD y FreeBSD).

Pero su naturaleza como producto de código abierto no le limita a estar disponible únicamente bajo este tipo de sistemas operativos. Snort puede funcionar bajo soluciones comerciales como, por ejemplo, Solaris, HP-UX, IRIX e incluso sistemas Microsoft Windows.

Desde el punto de vista del motor de detección, Snort estaría incluido en la categoría de detección basada en usos indebidos. Mediante un reconocimiento de firmas, Snort contrastará todo el tráfico capturado en sus reglas de detección.

Una regla de detección no es más que un conjunto de requisitos que le permitirán, en caso de cumplirse, activar una alarma. Por ejemplo, una regla de Snort que permitiría verificar el uso de aplicaciones *peer-to-peer* para el intercambio de ficheros a través de Internet verificaría el uso de la cadena `GET` en servicios diferentes al puerto tradicional del protocolo HTTP. Si un paquete capturado por Snort coincide con esta sencilla regla, su sistema de notificación lanzará una alerta indicando lo sucedido. Una vez que la alerta sea lanzada, puede ser almacenada de distintas maneras y con distintos formatos como, por ejemplo, un simple fichero de registro del sistema, una entrada en una base de datos de alertas, un evento SNMP, etc.

A continuación veremos los orígenes de Snort, así como un análisis de su arquitectura y algunas de sus características más destacables. Veremos también algunos problemas de seguridad que existen tras la utilización de Snort y la forma de solventarlos.

Versión comercial de Snort

Aunque Snort está disponible bajo licencia GPL (GNU Public License), existen productos comerciales basados directamente en Snort y distribuidos por la empresa Sourcefire, fundada por el creador de Snort, Marty Roesch. El análisis de estas versiones comerciales quedan fuera del propósito de este material. Para más información, visitad `www.sourcefire.com`.

5.1.1. Origen de Snort

De forma muy resumida, podemos definir Snort como un *sniffer* de paquetes con funcionalidades adicionales para el registro de éstos, generación de alertas y un motor de detección basado en usos indebidos.

Snort fue desarrollado en 1998 bajo el nombre de APE. Su desarrollador, Marty Roesch, trataba de implementar un *sniffer* multiplataforma (aunque su desarrollo inicial se hizo para el sistema operativo GNU/Linux) que contara con diferentes opciones de clasificación y visualización de los paquetes capturados. Marty Roesch implementó Snort como una aplicación basada en la librería `libcap` (para el desarrollo de la captura de paquetes) lo cual garantizaba una gran portabilidad, tanto en la captura como en el formato del tráfico recogido.

Algo más que un *sniffer* ...

El autor de Snort trataba de indicar con este nombre que su aplicación era algo más que un *sniffer*. La palabra Snort significa en inglés una acción de inhalar o esnifar de forma más obsesiva y violenta. Además, Marty dijo en su momento que ya tenía demasiadas aplicaciones que se llamaran `a.out` y que todos los nombres populares para *sniffers*, llamados *TCP-something* ya estaban cogidos.

Snort empezó a distribuirse a través del sitio web *Packet Storm* (`http://www.packetstormsecurity.com`) el 22 de diciembre de 1998, contando únicamente con mil seiscientas líneas de código y un total de dos ficheros fuente. Por aquella época, el uso principal que le dio su autor era como analizador de sus conexiones de red a través de un cable módem y como *debugger* de las aplicaciones de red que estaba implementado.

El primer analizador de firmas desarrollado para Snort (también conocido como analizador de reglas por la comunidad de desarrollo de Snort) se añadió como nueva funcionalidad de la aplicación en enero de 1999. Esta nueva funcionalidad permitió que Snort comenzase a ser utilizado como detector de intrusiones.

En diciembre de 1999 apareció la versión 1.5 de Snort. En esta versión su autor decidió una nueva arquitectura basada en *plug-ins* que aún se conserva en las versiones actuales. A partir de esta versión, Marty Roesch abandonó la compañía donde trabajaba y se empezó a dedicar a tiempo completo a añadir nuevas funcionalidades para mejorar las capacidades de configuración y facilitar su uso en entornos más profesionales. Gracias a la gran aceptación que su IDS estaba obteniendo entre la comunidad de administradores, Marty pensó que era un buen momento para ofrecer su producto con un soporte para empresas, y obtuvo la financiación necesaria para fundar *Sourcefire**.

* En `www.sourcefire.com` encontrareis más información.

Sin embargo, Snort continúa siendo código libre y promete seguir siéndolo para siempre. La última versión disponible de Snort es la 2.1, la cual se presenta con más de setenta y cinco mil líneas de código y una restructuración total en cuanto al diseño original de su arquitectura inicial.

Aunque el soporte y desarrollo actual de Snort se hace desde *Sourcefire* de forma comercial, existe la versión libre bajo licencia GNU. Esta versión puede ser descargada libremente desde `www.snort.org`, permitiendo que cualquier usuario pueda disponer de soporte para las últimas versiones disponibles y las últimas actualizaciones de los ficheros de reglas para dichas versiones.

Actualmente, Snort cuenta un gran repertorio de accesorios que permiten reportar sus alertas y notificaciones en diferentes gestores de base de datos (como MySQL y Postgres) y un gran número de preprocesadores de tráfico que permiten poder analizar llamadas RPC y escaneo de puertos antes de que éstos sean contrastados con el conjunto de reglas asociado en busca de alertas.

Los conjuntos de reglas de Snort también han ido evolucionando a medida que la aplicación lo iba haciendo. El tamaño de los últimos conjuntos de reglas para la última versión Snort disponibles para descargar incrementa de forma similar a la velocidad de aparición de nuevos *exploits*. Estos ficheros de reglas se encuentran actualmente clasificados en distintas categorías como, por ejemplo, P2P, ataques de denegación de servicio, ataques contra servicios web, virus, tráfico pornográfico, etc.

Cada una de estas reglas están asociadas a un identificador único (sensor ID, SID), que permite reconocer y encontrar información acerca del ataque o mal uso detectado. Por ejemplo, el SID para el ataque `SSH banner attack` es el `1838`. Además, gracias al uso mayoritario de Snort entre la comunidad de administradores de red, otros productos de IDS han adoptado el formato de las reglas de Snort, así como la codificación utilizada para los volcados de los paquetes capturados (basada el `libcap`).

El soporte de estos ficheros de reglas aumenta a diario. De este modo, cualquier usuario de Snort, o de cualquier otro IDS con un formato de reglas compatible, podría crear sus propias reglas a medida que vayan apareciendo nuevos ataques y colaborar con la comunidad de desarrollo de Snort para mantener perfectamente actualizada su base de firmas.

5.1.2. Arquitectura de Snort

Snort proporciona un conjunto de características que lo hacen una herramienta de seguridad muy potente, entre las que destacan la captura del tráfico de red, el análisis y registro de los paquetes capturados y la detección de tráfico malicioso o deshonesto. Antes de nombrar con mayor detalle las características de Snort, es importante conocer y comprender su arquitectura.

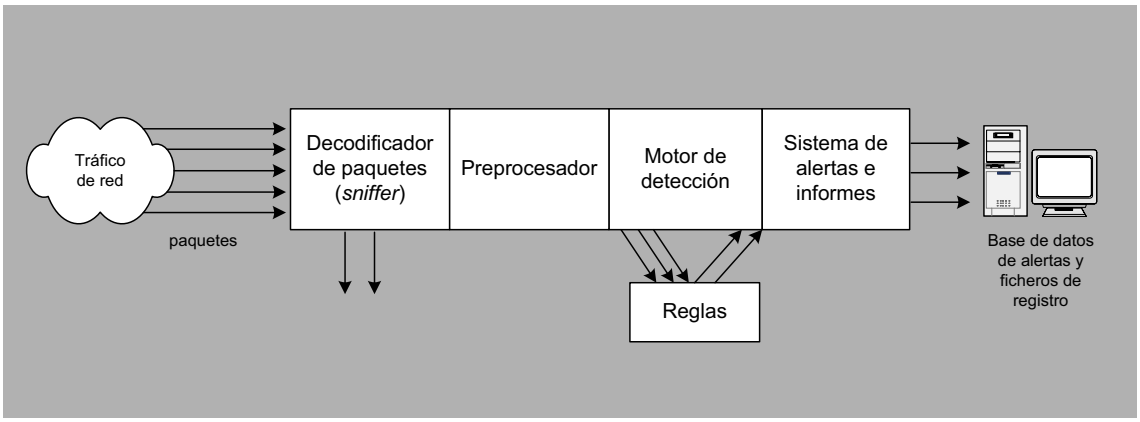
Snort está formado por un conjunto de componentes, la mayoría de los cuales son *plug-ins* que permiten la personalización de Snort. Entre estos componentes destacan los preprocesadores, que permiten que Snort manipule de forma más eficiente el contenido de los paquetes antes de pasarlos al elemento de detección, y su sistema de notificaciones y alertas basados en *plug-ins*, que permiten que la información reportada pueda ser enviada y almacenada en distintos formatos y siguiendo distintos métodos.

La arquitectura central de Snort se basa en los siguientes cuatro componentes:

- Decodificador de paquetes o *Sniffer*
- Preprocesador
- Motor de detección
- Sistema de alertas e informes

Siguiendo esta estructura, Snort permitirá la captura y el preprocesado del tráfico de la red a través de los dos primeros componentes (decodificador de paquetes y preprocesador), realizando posteriormente un chequeo contra ellos mediante el motor de detección (según el conjunto de reglas activadas) y generando, por parte del último de los componentes, las alertas y los informes necesarios.

La siguiente figura muestra la arquitectura básica de Snort que acabamos de comentar.



Observando la figura anterior podemos hacer un símil entre Snort y una máquina mecánica para la ordenación automática de monedas:

1) Toma todas las monedas (paquetes de la red recogidos por el decodificador de paquetes o *sniffer*).

2) Cada moneda se dejará caer por una rampa para determinar a qué grupo de monedas pertenece (preprocesador de paquetes).

3) Ordena las monedas según el tipo de moneda y las enrolla en forma de canutos según la categoría (motor de detección).

4) Finalmente, el administrador decidirá qué hacer con cada uno de los canutos de monedas ordenadas (sistema de alertas).

Tanto el preprocesador como el motor de detección y los componentes para la notificación de alertas son todos *plug-ins* de Snort. Un *plug-in* es una aplicación desarrollada conforme la API de *plug-ins* de Snort. Estas aplicaciones son utilizadas junto con el núcleo del código de Snort, pero están separadas del mismo, de modo que un cambio en el núcleo de Snort no les afecte.

A continuación examinaremos con mayor detalle cada uno de los cuatro componentes básicos de Snort que acabamos de ver.

Decodificador de paquetes

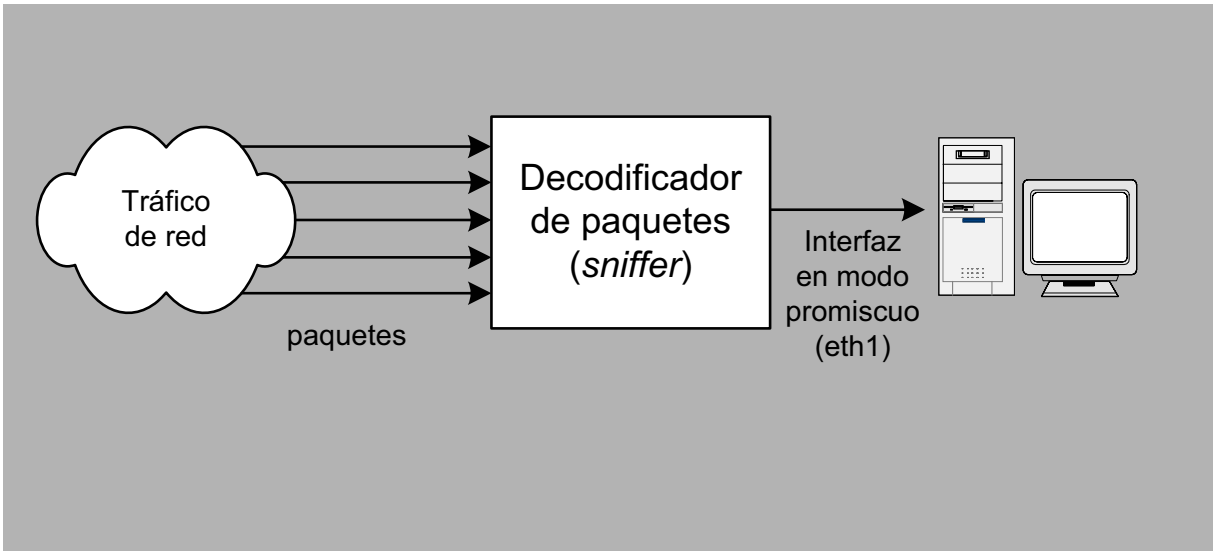
Un *sniffer* es un dispositivo (*software* o *hardware*) que se utiliza para poder capturar los paquetes que viajan por la red a la que es asociado.

En el caso de redes TCP/IP, este tráfico acostumbra a ser tráfico de datagramas IP, aunque también es posible la existencia de tráfico de distinto tipo como, por ejemplo, tráfico IPX o tráfico AppleTalk. Además, puesto que el tráfico IP consiste también en distintos tipos de protocolos, como TCP, UDP, ICMP, protocolos de encaminamiento, IPSec, ... muchos *sniffers* necesitarán conocer *a priori* el tipo de tráfico para poder interpretar más adelante los paquetes que van siendo recogidos y poder mostrarlos en un lenguaje comprensible por un administrador de red.

Como muchas otras herramientas relacionadas con la seguridad en redes, los *sniffers* pueden ser utilizados con objetivos más o menos deshonestos. Entre los distintos usos que se le puede dar a un *sniffer* podemos pensar en análisis de tráfico para la solución de congestiones y problemas de red, mejora y estudio del rendimiento de los recursos, captura pasiva de información sensible (contraseñas, nombres de usuario), etc.

Así, como el resto de *sniffers* tradicionales, el decodificador de paquetes de Snort será el elemento encargado de recoger los paquetes que más adelante serán examinados y clasificados por el resto de componentes. Para ello, el decodificador de paquetes deberá ser capaz de capturar todo aquel tráfico que le sea posible, para más adelante pasarlo al siguiente componente (el preprocesador) que se encargará de detectar qué tipo de tráfico se ha recogido.

En la siguiente figura vemos un esquema del funcionamiento del decodificador de paquetes de Snort.



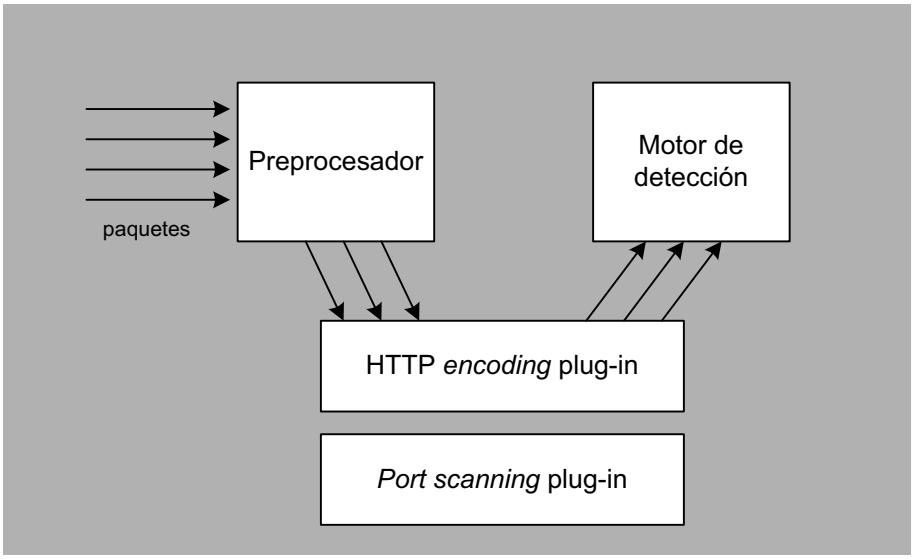
Preprocesador

A medida que el decodificador de paquetes vaya recogiendo el tráfico que pasa por la red, lo irá entregando al elemento de preprocesado para que lo vaya adaptando y se lo vaya entregando al motor de detección.

Así pues, el preprocesador irá obteniendo paquetes sin tratar (*raw paquets*) y los verificará mediante un conjunto de *plug-ins* (como, por ejemplo, el *plug-in* para llamadas RPC o el *plug-in* de escaneo de puertos). Estos *plug-ins* verificarán los paquetes en busca de ciertos comportamientos en éstos que le permita determinar su tipo. Una vez determinado el comportamiento del paquete, éste será enviado hacia el motor de detección.

Esta característica de preprocesamiento es realmente importante para una herramienta de detección, ya que es posible la utilización de terceras aplicaciones (en forma de *plug-ins*) que pueden ser activadas y desactivadas según las necesidades del nivel de preprocesado. Por ejemplo, si a un administrador de red no le preocupa el tráfico RPC que entra y sale de su red (y no necesita, por tanto, analizarlo) por cualquier motivo, no tendrá más que desactivar el *plug-in* de RPC y seguir utilizando el resto.

En la siguiente figura vemos un esquema donde el preprocesador de Snort utiliza dos de sus *plug-ins* para verificar el tipo del paquete que va recibiendo y pasarlo posteriormente al motor de detección.



Motor de detección

El motor de detección es el corazón de Snort desde el punto de vista de sistema de detección de intrusos. A partir de la información proporcionada por el preprocesador y sus *plug-ins* asociados, el motor de detección contrastará estos datos con su base de reglas. Si alguna de las reglas coincide con la información obtenida, el motor de detección se encargará de avisar al sistema de alertas indicando la regla que ha saltado.

Como ya adalantabamos, el motor de detección de Snort se basa en una detección de usos indebidos a través de un reconocimiento de firmas de ataque. Para ello, el motor de detección hace uso de los conjuntos de reglas asociados a Snort. Estos conjuntos de reglas están agrupados por categorías (troyanos, *buffer overflows*, ataques contra servicios web, etc.) y deben ser actualizados a menudo.

Una regla puede estar dividida en dos partes. En primer lugar tenemos la cabecera de la regla, en la que indicamos la acción asociada a ésta en caso de cumplirse (generación de un fichero de registro o generación de una alerta), el tipo de paquete (TCP, UDP, ICMP, etc.), la dirección de origen y destino del paquete, etc. En segundo lugar está el campo `option` de la regla, donde encontraremos la información que debe contener el paquete (en la parte de datos, por ejemplo) para que se cumpla la regla.

Snort posee una sintaxis propia para la creación de las reglas. Esta sintaxis incluye el tipo de protocolo, el contenido, la longitud, la cabecera, etc., que permiten especificar hasta el más minimo detalle de la condición que ha de darse para que un paquete cumpla dicha regla.

Éste es un ejemplo de regla de Snort:

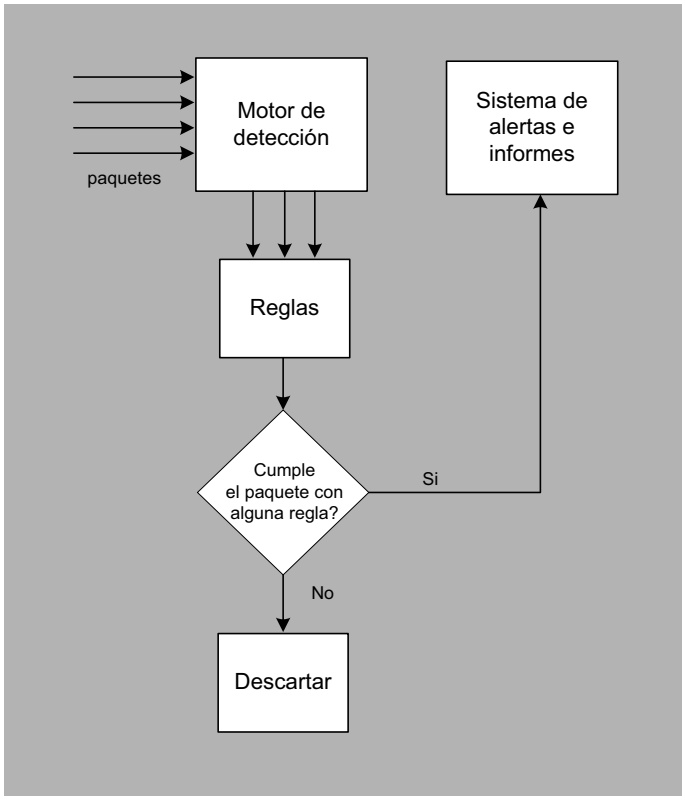
```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT STAT *
dos attempt"; flow:to_server,established; content:"STAT "; nocase;
content:"*"; reference:bugtraq,4482; classtype:attempted-dos;
sid:1777; rev:1;)
```

De todos los elementos que hemos visto, el motor de detección y la sintaxis utilizada por las reglas de detección son las partes más complicadas de comprender a la hora de estudiar el comportamiento de Snort.

Aun así, una vez que nos pongamos a trabajar con Snort y hayamos aprendido mínimamente la sintaxis utilizada, es bastante sencillo llegar a personalizar y ajustar el comportamiento de la funcionalidad de detección de Snort.

Además, los conjuntos de reglas pueden ser activados o desactivados con facilidad para poder así definir el comportamiento de detección deseado según el tipo de red donde Snort va a ser configurado.

En la siguiente figura podemos ver un sencillo esquema sobre el comportamiento general del motor de detección de Snort.



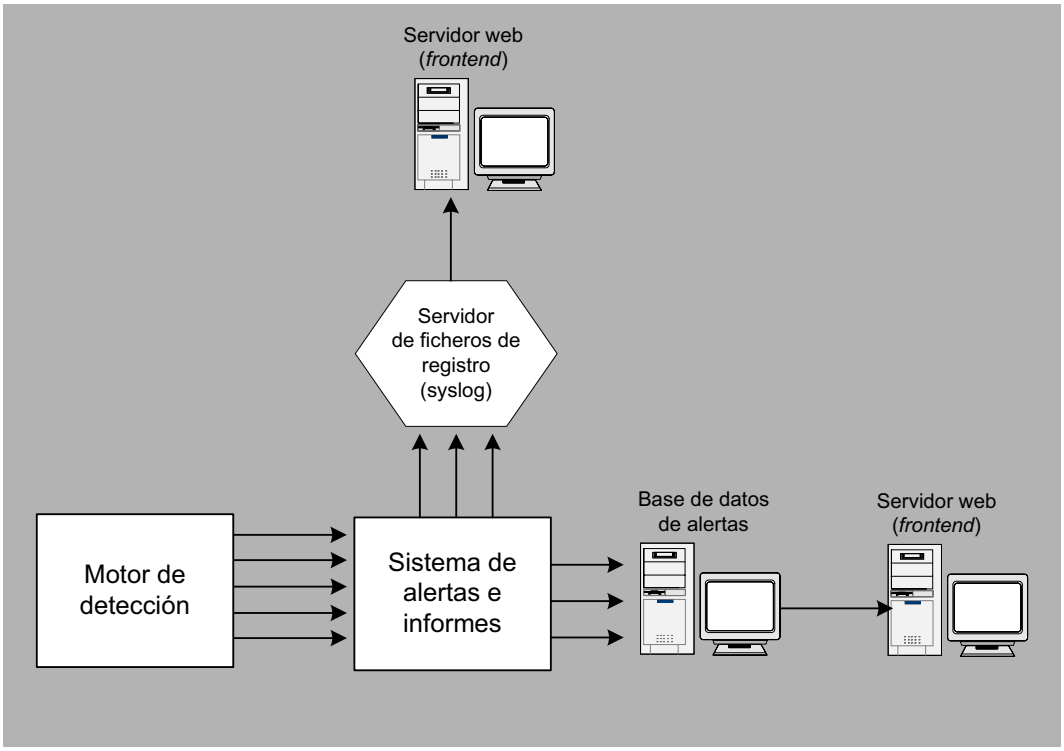
Sistema de alertas e informes

Una vez que la información capturada por el decodificador de paquetes de Snort es analizada por el motor de detección, los resultados deben ser reportados de alguna forma. Mediante este componente será posible realizar esta función, pudiendo generar los resultados en distintos formatos y hacia distintos equipos.

Cuando una alerta es lanzada por el motor de detección, esta alerta puede suponer la generación de un fichero de registro (*log*), ser enviada a través de la red mediante un mensaje SNMP o incluso ser almacenada de forma estructurada por algún sistema gestor de base de datos como, por ejemplo, MySQL o Postgres.

Además, es posible la utilización de herramientas alternativas (desarrolladas por terceras partes) que facilitan la visualización y el tratamiento de la información reportada por Snort. La mayor parte de estas herramientas están disponibles para poder ser descargadas libremente de Internet.

Como en el caso del motor de detección y el preprocesador, el sistema de alertas e informes de Snort también utiliza un esquema de *plug-ins* para el envío de alertas y notificaciones. En la siguiente figura podemos ver un esquema sobre cómo funciona este sistema de notificaciones a través de *plug-ins*.



5.1.3. Consideraciones de seguridad con Snort

Aunque la finalidad de una aplicación como Snort es la de mejorar la seguridad de nuestra red, es muy posible que este tipo de herramientas presenten vulnerabilidades (en su diseño, en su código, en su configuración, etc.), por lo que su instalación en una red de producción puede suponer un nuevo agujero en su seguridad.

Si un atacante consigue hacerse con un sistema donde está funcionando Snort, no será posible poder confiar en las alertas y notificaciones que este sistema de detección nos está ofreciendo. Será necesario reinstalar todo el sistema y configurarlo de nuevo para poder volver a confiar en él.

Imaginemos, por ejemplo, que el sistema de detección que hemos instalado en la red y donde está funcionando Snort tiene un puerto de SSH abierto para ofrecer la posibilidad de trabajo remoto contra dicho sistema. Además, el sistema almacena las alertas en una base de datos local y ejecuta un servidor de HTTP y HTTPS para ofrecer una interfaz web segura hacia las alertas y notificaciones reportadas por Snort y almacenadas en la base de datos local.

En este supuesto, el sistema de detección que hemos instalado en la red es igual de vulnerable (o incluso más) que cualquiera de los sistemas que queremos proteger. Para empezar, este sistema de detección tiene abiertos toda una serie de puertos TCP, por ejemplo: SSH (puerto 22), HTTP (puerto 80), HTTPS (puerto 443) y posiblemente MySQL (puerto 3306) o Postgres (puerto 5432). La mayor parte de los servidores que ofrecen estos servicios presentan deficiencias de seguridad y, dependiendo de las versiones y de las condiciones de configuración, es posible que puedan ser explotados para realizar un ataque de intrusión.

En ese momento, cualquiera que tenga acceso a dicha red podrá realizar un escaneo de puertos con alguna herramienta como, por ejemplo, Nmap, y tratar de realizar una captura de paquetes en caso de disponer de un equipo donde pueda colocar la tarjeta de red en modo promiscuo (o realizar un envenenamiento de ARP). Supongamos que dicho atacante encuentra, entre la información obtenida, que existen deficiencias en alguno de los servicios anteriores y logra realizar con éxito un ataque de intrusión en el sistema donde Snort está funcionando. En este caso, el sistema de detección dejará de ofrecer información de confianza y sus alertas carecerán de valor.

Aparte de las deficiencias de seguridad que pudiesen existir en los servidores del ejemplo anterior, también es posible encontrar este tipo de deficiencias en el propio código de Snort. Aunque en la corta existencia de esta aplicación el número de incidentes de seguridad reportados por la comunidad de desarrolladores y de usuarios no es demasiado elevada (seguramente gracias a su diseño minimalista y basado en cadenas de *plug-ins*), es posible encontrar versiones de Snort que presenten deficiencias de programación que puedan ser explotadas bajo ciertas condiciones.

Por el momento, algunas de las deficiencias reportadas contra versiones antiguas de Snort cuentan con la posibilidad de poder realizar una denegación de servicio contra el núcleo de la aplicación (dependiendo de cómo haya sido configurado) y algunas deficiencias de desbordamientos de *buffer* relacionadas con algunos de los *plug-ins* desarrollados por terceras partes. Aun así, y a diferencia de muchas otras herramientas de seguridad, Snort cuenta con un historial de deficiencias de seguridad realmente muy bajo.

5.1.4. Utilización de ACID como interfaz gráfica

Alertas de Snort

Si tratáis de poner en marcha Snort en una red congestionada y con un número razonable de firmas de detección activadas, la cantidad de alertas lanzadas por Snort puede alcanzar la centena en poco tiempo. Este enorme volúmen de información no será fácil de tratar con la utilización de un simple navegador de ficheros de registro.

El propósito de instalar Snort en una red no es únicamente obtener información (intentos de intrusión), sino analizar tal información y poder tomar las acciones necesarias en función de los datos obtenidos. Si el número de reglas activadas es elevado y el tráfico de la red aumenta con facilidad, no será del todo sencillo analizar la información reportada por Snort a no ser que utilicemos herramientas de visualización adecuadas.

Por otro lado, la faceta interesante de un sistema de detección como Snort no es simplemente registrar eventos o alertas, sino ser capaz de reaccionar a los intentos de intrusión en un periodo de tiempo razonablemente corto. Así pues, será necesario el uso de segundas aplicaciones que ayuden a consolidar y analizar la información reportada por Snort, con el objetivo de alertar a los administradores de la red de los intentos de intrusión analizados.

Actualmente, existe un gran número de utilidades para trabajar con las alertas generadas por Snort. Algunas de estas herramientas son mantenidas por la propia comunidad de desarrolladores de Snort, aunque también podemos encontrar aplicaciones desarrolladas por terceras partes. Éste es el caso de la interfaz *ACID* (*Analysis Console for Intrusion Databases*) desarrollada dentro del proyecto *AIRCERT* del centro de coordinación CERT de *Carnegie Mellon*. En el momento de escribir este documento, ACID es probablemente la mejor solución basada en *software* libre para el análisis de las alertas y eventos reportados por Snort.

Roman Danyliw

Aunque dentro del proyecto colaboran distintas personas, el código de ACID sigue siendo mantenido por su creador original, Roman Danyliw. ACID puede ser descargado libremente desde la url `http://acid-lab.sourceforge.net/`

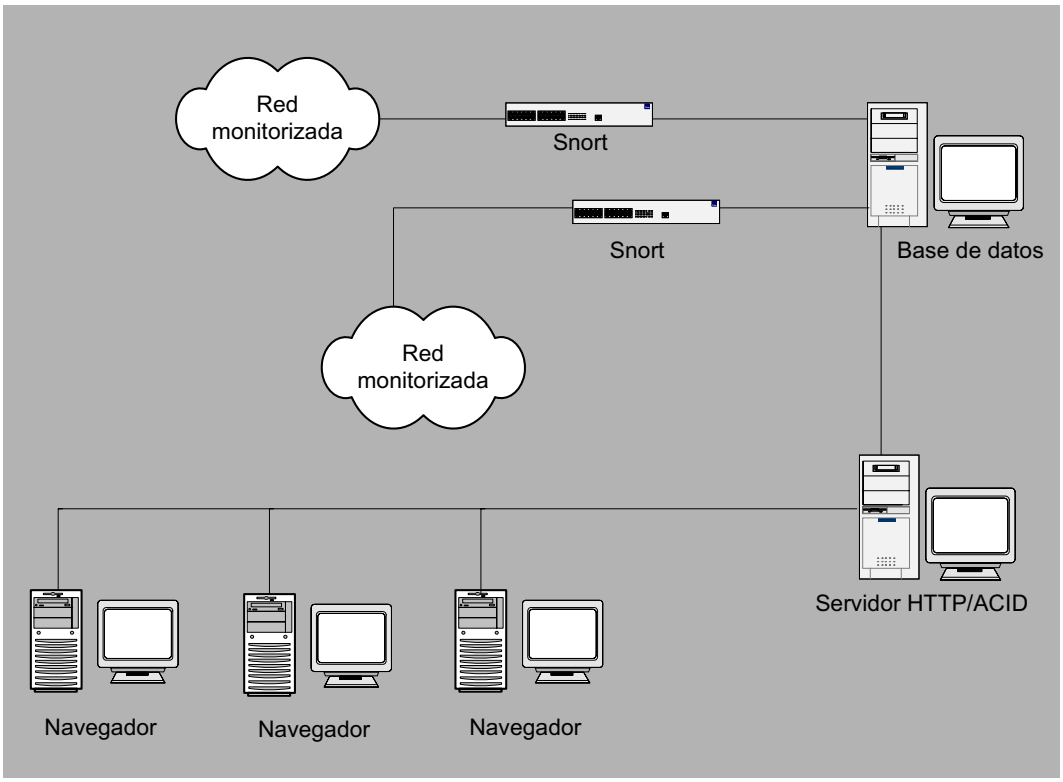
ACID es básicamente un conjunto de *scripts* escritos en PHP que proporcionan una interfaz entre un navegador web y la base de datos donde Snort irá almacenando las alertas. Aunque se trata de una herramienta aún en construcción, su desarrollo cuenta ya con más de cuatro años de experiencia. Durante este tiempo, ACID se ha convertido en una herramienta muy potente para la consolidación y el análisis de alertas generadas por Snort. Aunque inicialmente ACID fue pensado para procesar únicamente información reportada por Snort, actualmente ACID es independiente de la base de datos de Snort y puede procesar información de otros productos. Por ejemplo, es posible combinar ACID para analizar información reportada a través de *netfilter*, o mensajes de control de acceso generados por productos de Cisco. Otras de las características que podemos destacar de ACID son las siguientes:

- Decodificación y visualización de paquetes TCP/IP.
- Creación de diagramas y estadísticas basadas en fechas, horas, firmas, protocolo, etc.

- Interfaz para la realización de búsquedas y creación de consultas. Los resultados de estas acciones se devolverán de forma estructurada con información para facilitar la comprensión de las alertas lanzadas por Snort. En esta información se resaltarán las direcciones de origen/destino, los puertos de origen/destino, estado de las banderas TCP/IP (*TCP/IP flags*), etc.

- Gestión de alarmas. Se proporciona la posibilidad de crear grupos de alarmas lógicos, donde almacenar la información de los incidentes que sean necesario destacar. También existen opciones de manejo de las alarmas, permitiendo la eliminación de falsos positivos, exportación a través de correo electrónico y/o almacenamiento de las alertas encontradas en la base de datos.

La estructura de ACID es multinivel y fácilmente escalable. Puede ser utilizado tanto en un sistema aislado de la red, como con distintos equipos repartidos en diferentes capas de la red. La siguiente figura muestra la parte lógica del sistema:

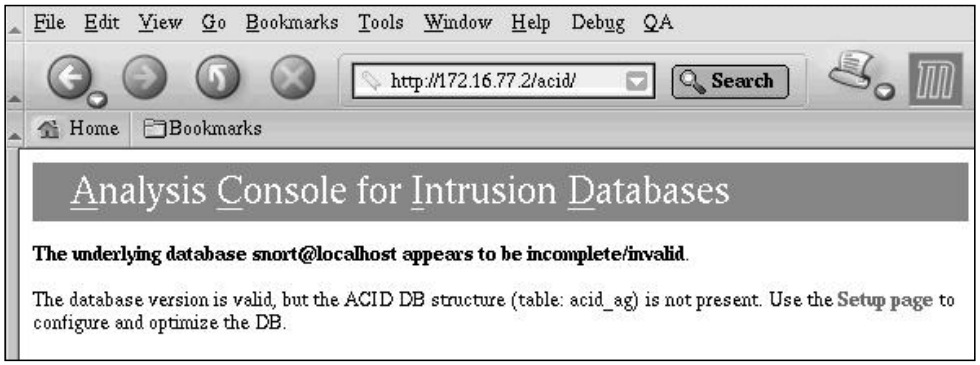


Como se puede ver en la figura anterior, ACID trabaja con las alertas que diferentes sensores de Snort irán depositando en una base de datos. Un conjunto de *scripts* escritos en lenguaje PHP se encargarán de realizar las consultas pertinentes a la base de datos y transformar los resultados en HTML para poder navegar desde un cliente de HTTP por los resultados ofrecidos por ACID. Los SGBD soportados por ACID son oficialmente PostgreSQL y MySQL, aunque es posible modificar el código de la aplicación para poder trabajar con otros SGBD basados en SQL y soportados por PHP. Respecto al servidor de HTTP, ACID puede funcionar correctamente bajo cualquier servidor web que soporte PHP4. Aun así, es posible encontrarse con complicaciones con según qué servidores a causa de las restricci-

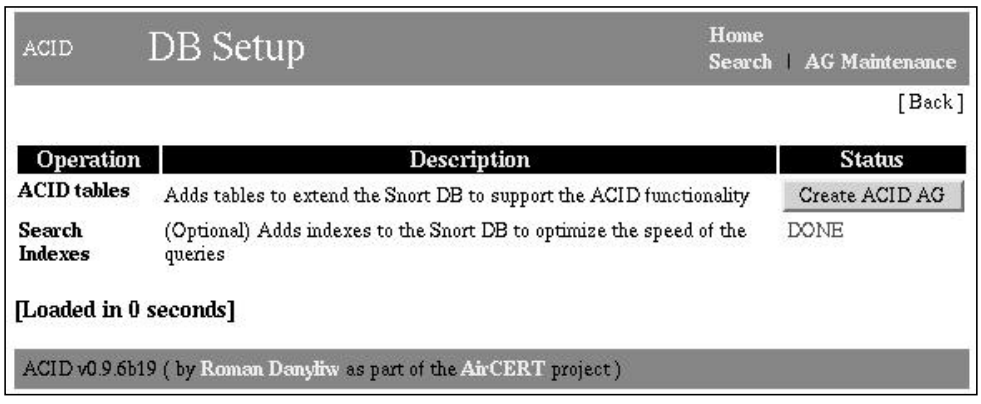
ones relacionadas con la generación de gráficas y estadísticas. Esta parte de la aplicación está especialmente pensada para funcionar en un sistema GNU/Linux, junto con el servidor de HTTP Apache.

La instalación y configuración de los prerrequisitos básicos de ACID en un sistema GNU/Linux (conjunto de sensores de Snort, servidor de HTTP Apache, intérprete de lenguaje PHP4, librerías gráficas necesarias, SGBD, etc.) no serán detallados en el presente documento, pues escapan a la finalidad del mismo. Así pues, veremos a continuación un ejemplo de cómo utilizar ACID en un sistema con los anteriores prerrequisitos ya instalados y configurados.

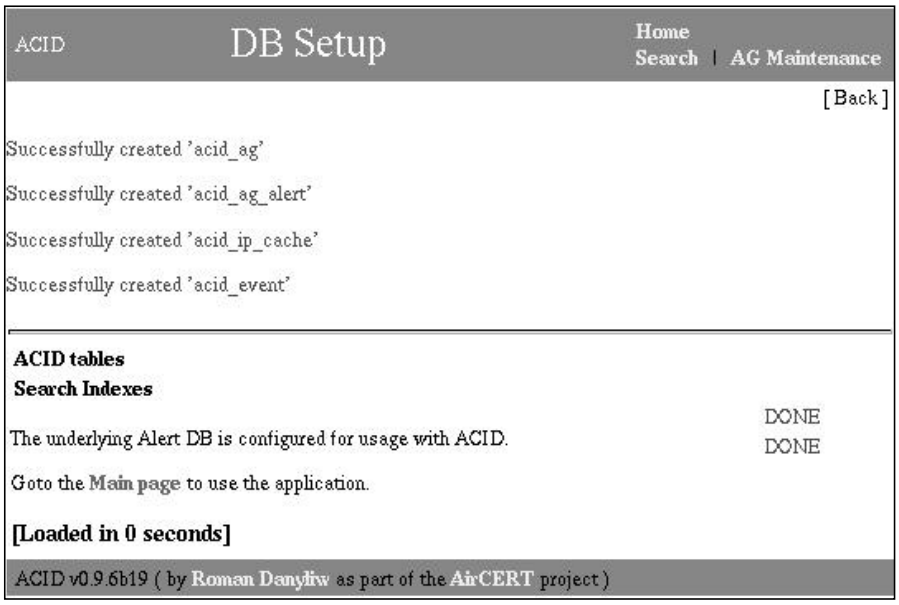
Suponiendo que tenemos ACID correctamente instalado en un sistema GNU/Linux con dirección IP `172.16.77.2`, trataremos de acceder a la interfaz principal de ACID desde un cliente de HTTP a través de la URL `http://172.16.77.2/acid/`. Si es la primera vez que accedemos a ACID, nos aparecerá en el navegador la siguiente pantalla:



El motivo de la pantalla anterior no es más que anunciarnos que existen algunas tablas de la base de datos que aún no han sido generadas. Normalmente, la base de datos que utilizará Snort viene en un fichero de consultas SQL para un SGBD MySQL o PostgreSQL. Pero en dicho fichero no vienen las tablas que utilizará ACID para indexar las tablas de `Snort`. Si pulsamos sobre el enlace `Setup page`, el servidor ejecutará el *script* necesario para generar las tablas requeridas.

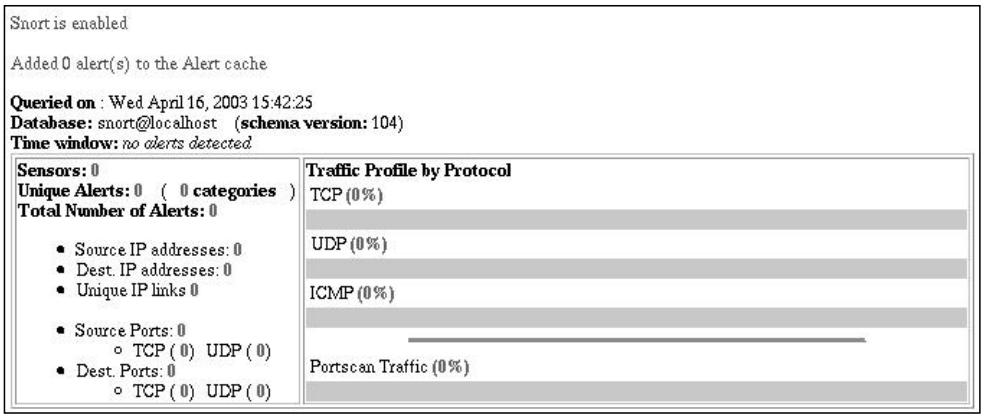


Si todo va bien, es decir, el guión se ejecuta correctamente en el servidor y las tablas necesarias para el funcionamiento de ACID se crean correctamente, aparecerá en nuestro navegador la siguiente pantalla:



A partir de este momento, podemos empezar a utilizar ACID. Como veremos en las siguientes imágenes, su utilización es muy sencilla. Nada más entrar, veremos la pantalla principal de ACID. Parte de la información que veremos en esta pantalla son las estadísticas generales de la actividad reportada con Snort: el nombre y la categoría de las alertas que han sido lanzadas, el número de alertas agrupado por protocolos, porcentajes de alertas lanzadas, etc.

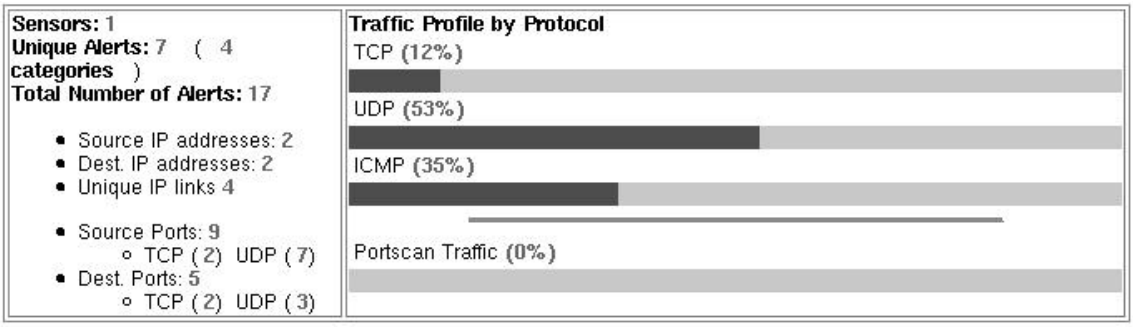
Cada una de estas informaciones se ofrece en forma de enlace. Pulsando sobre cualquiera de estos enlaces, podremos entrar y seleccionar la información correspondiente. Por el momento, la única información reportada es el nombre de la base de datos (`snort@localhost`), la marca de tiempo de la última consulta realizada (`16 de abril de 2003`) y la versión del esquema utilizado. Por el momento, y tal como indica la etiqueta `time window`, ninguna alerta habrá sido reportada.



Para comprobar el correcto funcionamiento de Snort, y asegurarnos de que ACID accede sin problemas a las alertas reportadas en la base de datos, realizaremos desde algún otro equipo con acceso a la red local una exploración de puertos mediante la herramienta Nmap:

```
root$nmap 172.16.77.2

22/tcp      open
111/tcp     open
993/tcp     open
995/tcp     open
1024/tcp    open
```

Una vez realizada la exploración de puertos anterior, veremos cómo, al refrescar la pantalla principal, ACID nos informa de que cuatro alertas han sido añadidas a la base de datos de Snort.



En la figura anterior podemos ver cómo el resto de marcadores de la pantalla principal se han actualizado. La información más importante a destacar es que la base de datos recibe alertas de un único sensor, que el número total de alertas reportadas es de cuatro y que el `time window` es tan sólo de un segundo. Por otro lado, de las cuatro alertas, vemos que hay tres alertas únicas, organizadas en un total de dos categorías.

También podemos observar en la figura anterior que dos de las alertas pertenecen a tráfico TCP, y otras dos pertenecen a tráfico ICMP. Respecto a las estadísticas sobre direcciones IP y puertos TCP y UDP, podemos contrastar también que dos direcciones IP de origen, dos direcciones IP de destino, y cuatro puertos TCP han sido reportados.

Para elevar un poco más el número de alertas reportadas por `Snort`, podemos utilizar a continuación algún escáner de vulnerabilidades contra el equipo que alberga el sensor de Snort, o contra algún equipo que se encuentre dentro de la misma red. El escáner de vulnerabilidades escogido es `Raccess`.

Raccess (*Remote Access Session*) es un escáner de vulnerabilidades en red que analiza la integridad de un sistema trantando de obtener un acceso ilegal contra alguno de sus equipos mediante el uso de técnicas remotas de intrusión.

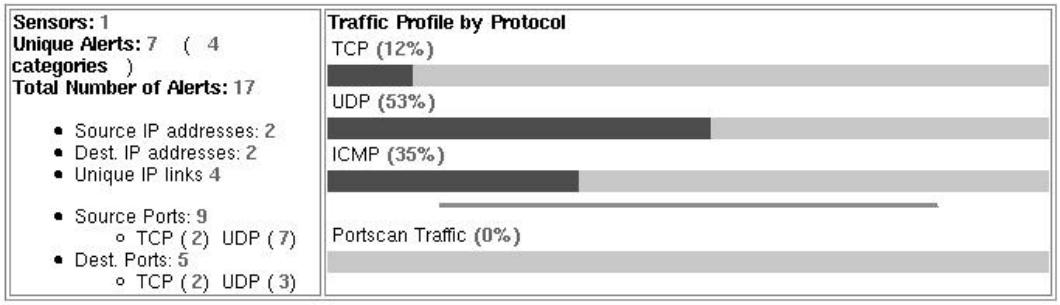
La principal diferencia entre Raccess y otros escáneres de vulnerabilidades similares como, por ejemplo, Nessus, es la utilización de ataques reales contra los equipos a analizar. De este modo, si Raccess encuentra una vulnerabilidad remota en alguno de los equipos analizados, tratará de obtener una cuenta con privilegios de administrador.

Actualmente, Raccess incluye un gran número de ataques remotos reales para distintos sistemas operativos y funciona sobre sistemas GNU/Linux, BSD y Solaris. Aun así, Raccess no es una herramienta para atacantes. Raccess es una utilidad para administradores y especialistas en seguridad de redes. De hecho, no utiliza técnicas silenciosas para tratar de pasar desapercibida. Por tanto, se trata de una herramienta muy ruidosa, cuya utilización es fácil de detectar por las máquinas remotas.

La siguiente imagen muestra la utilización de Raccess contra el equipo `172.16.77.2`, aunque los resultados ofrecidos por Raccess han sido simplificados. Como vemos, tras realizar una primera exploración de puertos contra el equipo elegido para la exploración, Raccess seleccionará de su base de datos de ataques aquellos que coincidan con los servicios abiertos, teniendo en cuenta el sistema operativo de destino, el servidor que ofrece cada servicio, etc.

```
root$raccess 172.16.77.2
...

--Service ssh Port 22: Opened!!--
Banner Info: SSH-1.99-OpenSSH_3.1p1

--Checking named version...
Named version: 8.2.2-P5

--Service sunrpc Port 111: Opened!!--

...

~Scan Completed!!

Starting attack session...

...
Named exploit
Do you want to run this exploit (y/n)? Y
...
Wu-ftpd exploit
Do you want to run this exploit (y/n)? Y
...
```

De nuevo en ACID, si pulsamos sobre la opción del navegador para refrescar la pantalla principal, o esperamos a que la interfaz se autorrefresque por sí sola, podremos ver cómo se han actualizado los índices tras la ejecución de algunos de los ataques de explotación ofrecidos por `raccess`.

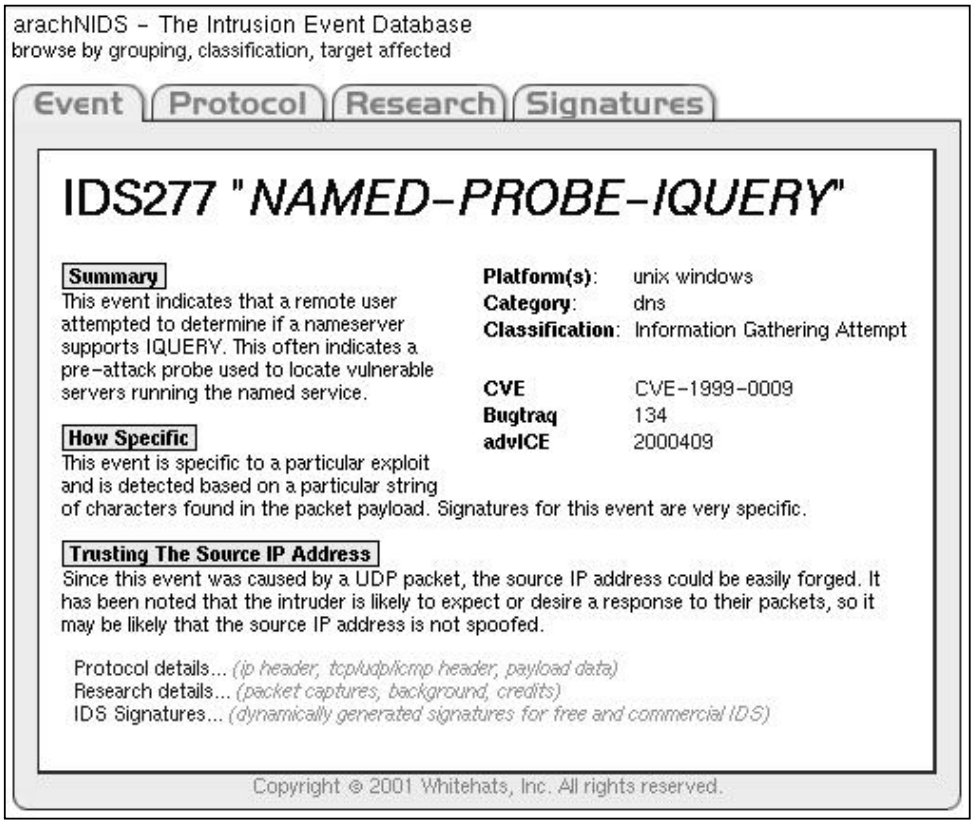


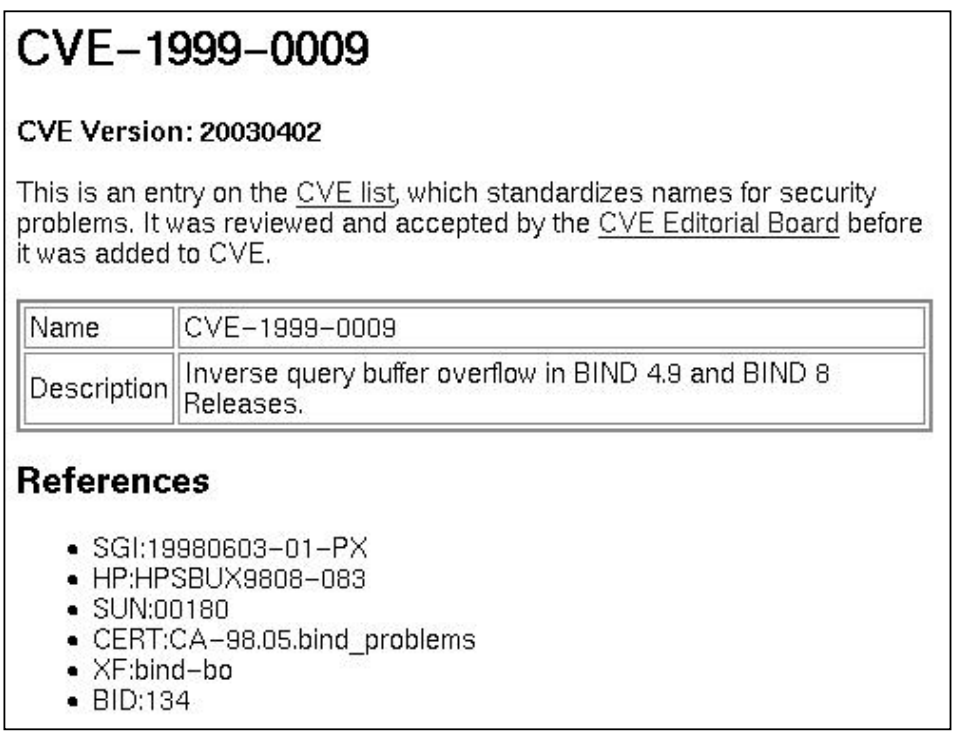
A continuación, pulsaremos sobre el enlace que apunta hacia la pantalla de alertas únicas (`Unique alerts`) para poder ver un listado de las siete alertas agrupadas por la categoría de la regla de detección que las ha hecho saltar. Podremos realizar una ordenación de las alertas, de forma ascendente o descendente, para, más adelante, visualizar únicamente aquellas alertas que se producen con mayor frecuencia. Para ello, basta con pulsar sobre la fecha correspondiente (sobre los símbolos > o <) en la cabecera de la columna deseada. La siguiente figura muestra la pantalla de alertas únicas.

	Signature	Classification	Total #	Sensor #	Src. Addr.	Dest. Addr.	First	Last
☐	[arachNIDS] MISC Large ICMP Packet	bad-unknown	6 (35%)	1	2	2	2003-03-25 16:38:07	2003-03-25 16:42:17
☐	INFO - Possible Squid Scan	attempted-recon	1 (6%)	1	1	1	2003-03-25 16:38:08	2003-03-25 16:38:08
☐	[arachNIDS] [CVE] [bugtraq] DNS named iquery attempt	attempted-recon	1 (6%)	1	1	1	2003-03-25 16:38:29	2003-03-25 16:38:29
☐	SCAN Proxy attempt	attempted-recon	1 (6%)	1	1	1	2003-03-25 16:38:30	2003-03-25 16:38:30
☐	[arachNIDS] RPC portmap request mountd	rpc-portmap-decode	2 (12%)	1	1	1	2003-03-25 16:41:59	2003-03-25 16:42:17
☐	[arachNIDS] RPC portmap request rstatd	rpc-portmap-decode	3 (18%)	1	1	1	2003-03-25 16:42:27	2003-03-25 16:42:37
☐	[arachNIDS] RPC EXPLOIT statdx	attempted-admin	3 (18%)	1	1	1	2003-03-25 16:42:27	2003-03-25 16:42:37

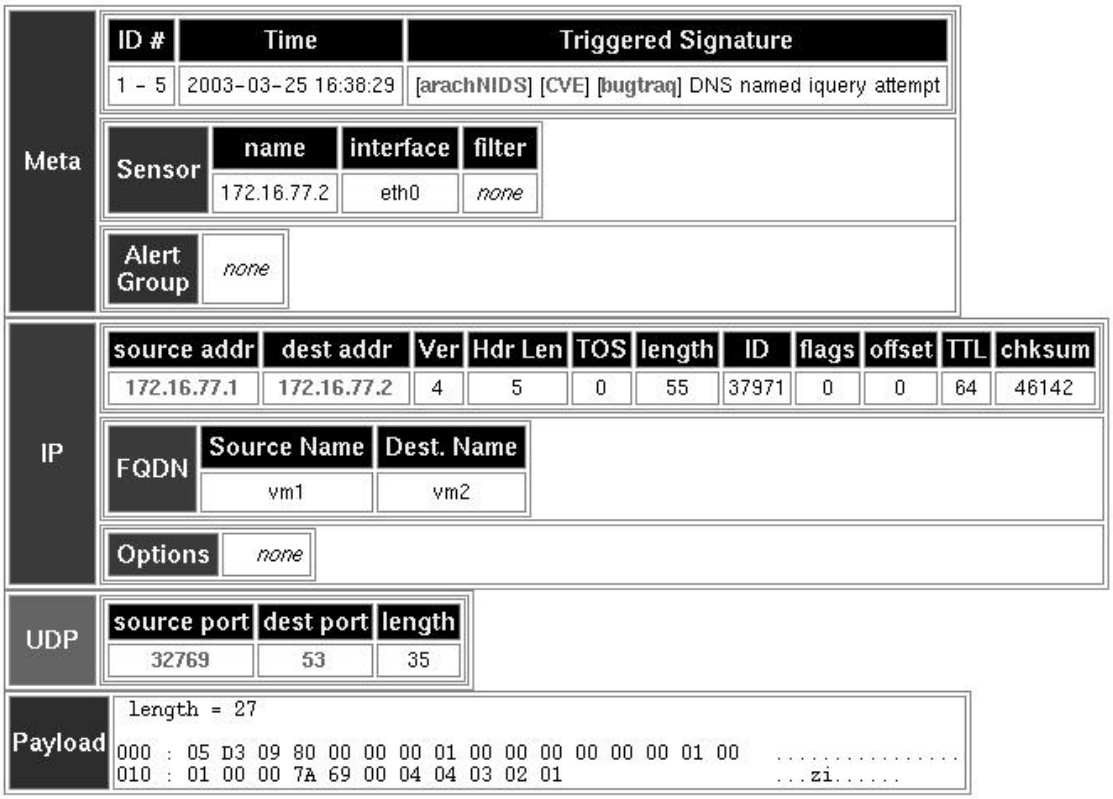
Si analizamos la pantalla anterior, vemos cómo cada línea muestra una de las alertas reportadas por Snort. Cada una de estas líneas consta de gran número de campos que podremos seleccionar. Dos de los campos más interesantes son el campo de clasificación de la alerta y la referencia hacia la base de datos de ataques (como, por ejemplo, *Arachnids* o *CVE*) que contendrá la explicación del ataque detectado. Esta información, así como la información relacionada con el resto de los campos, es obtenida de las reglas cuando Snort registra la alerta en la base de datos. Si pulsamos sobre el enlace que apunta hacia *Arachnids* o *CVE*,

podremos ver una explicación detallada de la alerta lanzada. El campo de clasificación, por otro lado, ayuda a agrupar las alertas por categorías. En la siguiente figura, podemos ver la información reportada por *Arachnids* respecto a uno de los ataques reportados por Snort tras ejecutar Racess.





Finalmente, si pulsamos sobre el identificador de alguna de las alertas, pasaremos a ver la información relacionada con el paquete que provocó la alerta. Como vemos en la siguiente figura, cada paquete registrado por Snort es mostrado por ACID de manera no codificada. Así, veremos la información relacionada con los flags TCP (si los hay), las opciones y el contenido del paquete, la dirección IP de origen y destino del datagrama IP, el *payload* del paquete, etc. Esta información puede ser de gran utilidad para determinar si efectivamente la alerta lanzada es una alerta real, o si por el contrario es una falsa alarma. En este caso, podremos realizar los ajustes necesarios sobre Snort para tratar de reducir el número de falsas alarmas.



Meta

ID #	Time	Triggered Signature
1 - 5	2003-03-25 16:38:29	[arachNIDS] [CVE] [bugtraq] DNS named iquery attempt

Sensor	name	interface	filter
	172.16.77.2	eth0	*none*

Alert Group	*none*

IP

source addr	dest addr	Ver	Hdr Len	TOS	length	ID	flags	offset	TTL	chksum
172.16.77.1	172.16.77.2	4	5	0	55	37971	0	0	64	46142

FQDN	Source Name	Dest. Name
	vm1	vm2

Options	*none*

UDP

source port	dest port	length
32769	53	35

Payload

```
length = 27

000 : 05 D3 09 80 00 00 00 01 00 00 00 00 00 00 01 00   ................
010 : 01 00 00 7A 69 00 04 04 03 02 01                  ...zi.....
```

Resumen

En este módulo hemos realizado una primera aproximación a Snort, una herramienta para la detección de intrusos basada en *software* libre. El objetivo principal de Snort es ayudar a los administradores de una red a realizar una vigilancia continuada del tráfico de la red en busca de intentos de intrusión o usos indebidos en la misma.

Al inicio del módulo hemos repasado brevemente los orígenes de la herramienta y hemos estudiado de modo general su arquitectura y su forma de trabajar. También hemos visto algunas de las deficiencias existentes en Snort, incluyendo como problemática la posibilidad de falsos positivos y falsos negativos por parte del producto.

Otra problemática a la hora de utilizar Snort, al igual que ocurre con otras herramientas similares, es la dificultad de navegar y analizar la gran cantidad de alertas o avisos de seguridad que se producirán. Generalmente, entre el tráfico normal de una red conectada a Internet encontraremos una elevada cantidad de intentos de intrusión o ataques en general. Así pues, si instalamos un sensor de Snort en una red de similares características, con un volumen de tráfico elevado, es muy probable que el número de alertas supere el millar en poco tiempo.

Actualmente, existe un gran número de aplicaciones basadas en *software* libre para realizar tareas de consolidación de información y análisis de las alertas de Snort. Estas aplicaciones serán un buen complemento para tratar de ajustar la base de firmas de Snort, reduciendo así el primero de los inconvenientes comentados, y ayudando en las tareas de análisis y consolidación para solucionar el segundo problema.

De las distintas soluciones existentes hemos visto ACID como interfaz de consultas y de navegación sobre las alertas de Snort. ACID es una herramienta interactiva con interesantes funcionalidades para la exploración y análisis de información. No se trata de una interfaz exclusiva para Snort, sino que puede ser utilizada como interfaz de otras herramientas similares. Desarrollada dentro del proyecto *AIRCERT* del centro de coordinación CERT de *Carnegie Mellon*, ACID o *Analysis Console for Intrusion Databases* es, en el momento de escribir esta documentación, la mejor solución basada en *software* libre para el análisis de las alertas y eventos reportados por Snort.

A lo largo de la última sección, hemos visto cómo ACID proporciona los medios oportunas para ejecutar las consultas oportunas a la base de datos de alertas. Se trata de una interfaz muy sencilla de utilizar, con un gran número de funcionalidades para facilitar el trabajo del analista como, por ejemplo, agrupación de alertas por categorías, visualización a alto nivel de los paquetes responsables de generar las alertas, estadísticas de las alertas por protocolo, etc. Otras funcionalidades, como la generación de informes y gráficas, son también fácilmente realizadas mediante la utilización de esta interfaz.

Bibliografía

[1] Beale, J.; Foster, J. C.; Posluns J. ; Caswell, B. (2003). *Snort 2.0 Intrusion Detection.* Syngress Publishing.

[2] Eyler, P. (2001). *Networking Linux: A Practical Guide to TCP/IP.* New Riders Publishing.

[3] Rehman, R. (2003). *Intrusion Detection Systems with Snort. Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID.* Prentice Hall PTR.

[4] Stanger, J.; Lane, P. T.; Danielyan E. (2001). *Hack Proofing Linux: A Guide to Open Source Security.* Syngress Publishing, Inc.

[5] Toxen, B. (2000). *Real World Linux Security: Intrusion Prevention, Detection, and Recovery.* Prentice Hall PTR.