OS Fingerprinting

Jaime Pérez Crespo <jperez@blackspiral.org>

10 de Junio del 2006

Resumen

Uno de los factores más importantes a la hora de realizar un ataque informático consiste en conocer la mayor cantidad de detalles posible sobre el sistema objetivo. Para el administrador de sistemas, evitar dar a conocer detalles sobre el servicio puede servir de ayuda para mantenerlo seguro. En el presente artículo veremos las técnicas desarrolladas hasta el día de hoy para averiguar el sistema operativo que se ejecuta en una máquina remota sin necesidad de tener acceso a la misma. Se tratarán además diversos métodos de los que dispone el adminsitrador para evitar este tipo de problemas y modificar la apariencia de su sistema hacia el exterior.

1. Introducción

La técnica del *fingerprinting* de sistema operativo consiste en la suposición del sistema operativo y su versión en base al análisis de la pila de comunicaciones de una máquina. Se basa, como su nombre indica, en la obtención de una serie de patrones de comportamiento observados en los paquetes que genera para formar una "huella dactilar" que lo identifique unívocamente. Como es lógico, este tipo de información facilita sobremanera cualquier tipo de ataque sobre una máquina remota, e incluso en ocasiones se hace imprescindible.

Antiguamente la información relativa al sistema se obtenía de forma muy ingenua consultando los *banners* o anuncios de los servicios ejecutándose en una máquina. Un *banner* es una cadena informativa que se muestra al conectarse a un servicio remoto, que puede llegar a dar todo tipo de detalles. Incluso algunos servicios como FTP proporcionan comandos como "*SYST*" para obtener este tipo de información. A modo de ejemplo, comprobemos qué tipo de información podemos obtener de un servidor simplemente conectando a su servicio de correo y su servicio web:

```
% telnet dac.escet.urjc.es 25
Trying 212.128.1.79...
Connected to dac.escet.urjc.es.
Escape character is '^]'.
220 dac.escet.urjc.es ESMTP Sendmail
8.13.4/8.13.4/Debian-3; Sat, 10 Jun
2006 17:35:52 +0200; (No UCE/UBE)

% echo 'GET / HTTP/1.0\n' |
nc dac.escet.urjc.es 80 |
egrep '^Server:'
Server: Apache/2.0.54 (Ubuntu) PHP/5.0.5-2
ubuntu1.2 mod_ssl/2.0.54 OpenSSL/0.9.7g

% echo 'GET / HTTP/1.0\n' |
nc kybele.escet.urjc.es 80 |
egrep '^Server:'
Server: Microsoft-IIS/5.0
```

No hace falta demasiada imaginación para saber qué sistema operativo ejecutan los dos servidores mostrados en el ejemplo, incluso precisando la versión del mismo y de sus servicios. Este tipo de datos son vitales a la hora de buscar vulnerabilidades que aprovechar para atacar una máquina, por lo que si bien, la seguridad por obscuridad no es una buena política, el hecho de que sea posible averiguar esta información por otras vías no significa que el administrador facilite de ningún modo su obtención.

A día de hoy esto deja de ser la tónica general. La mayoría de administradores de sistemas configuran

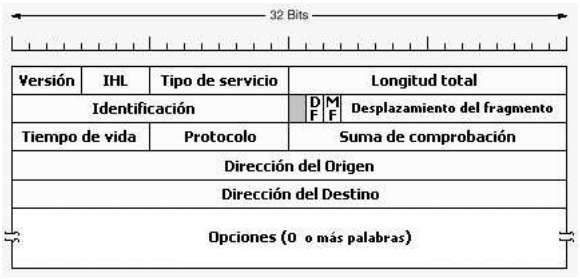
sus servicios para que no notifiquen este tipo de información en los anuncios, o bien muestren una falsa. Es por ello que se deben utilizar técnicas más sofisticadas como las descritas en este artículo y que se engloban bajo el nombre de "*fingerprinting*". Dicha técnica consiste, a grandes rasgos, en obtener paquetes provenientes de la máquina objetivo, analizar sus características y compararlos con una base de datos de "peculiaridades" para poder identificar el sistema operativo que los generó. Dado que los estándares definidos en las *RFC's* de TCP y otros protocolos dejan partes concretas abiertas a la implementación, es posible distinguir distintos sistemas operativos en base a la interpretación que hacen de la norma. De hecho, algunos tales como Microsoft Windows, las contradicen de forma explícita.

Atendiendo a la forma en la que se obtienen paquetes que analizar se puede distinguir entre dos métodos de *fingerprinting*, **activo** y **pasivo**.

2. Fingerprinting pasivo

Cuando se utiliza la variante pasiva de la técnica del fingerprinting los paquetes a analizar se obtienen directamente de la red local, lo cual quiere decir que el sistema atacante no genera ningún tipo de comunicación hacia el destino con el fin de provocar una respuesta. La implicación inmediata de esta técnica es que no permite analizar sistemas fuera de la red local, y que obliga al atacante a configurar su dispositivo de red en modo *promíscuo* [1]. Adicionalmente significa que el atacante está a merced del tráfico que circule por la red. Si desea obtener información de una máquina concreta es posible que tenga que esperar bastante tiempo o que incluso nunca llegue a ver un paquete proveniente de dicha máquina.

Por lo general se analizan dos cabeceras distintas en busca de signos diferenciadores [2]: la cabecera IP de nivel de red y la cabecera TCP de nivel de transporte [8]. La siguiente figura muestra un ejemplo de cabecera IP:



Uno de los campos más interesantes de la cabecera IP es el del "Tipo de Servicio" o *TOS*. Indica si los datos transmitidos tienen algún tipo de requisito. La mayoría de sistemas operativos lo mantienen al valor 0x00 por defecto, mientras que otros como OpenBSD lo inicializan a 0x08 (minimizar el retardo) cuando se abren conexiones con protocolos como telnet. Las posibles opciones del campo TOS son las siguientes:

- `1000`: minimizar el retardo.
- `0100`: maximizar rendimiento.
- `0010`: maximizar fiabilidad.
- `0001`: minimizar el coste monetario.
- `0000`: servicio convencional.

Por tanto, atendiendo a las opciones encontradas en dicho campo podemos identificar o descartar de forma sencilla qué sistema operativo las ha producido. Por otra parte, la longitud total del paquete es quizás el campo más importante. Referencia la longitud de la cabecera embebida, de la cabecera IP y del relleno. La variación de este campo es muy alta entre distintos sistemas operativos, sobretodo en paquetes *SYN* de apertura de conexión. Cada sistema utiliza sus propias opciones de TCP, incluyendo *NOP's*, lo que hace que cada uno tenga una longitud distinta de datagrama. En general, podemos estar seguros de que cualquier paquete con menos de 44 bytes ha sido manipulado, ya que todos los sistemas operativos utilizan al menos una opción *MSS* de 4 bytes en el paquete *SYN*. Linux tiene una longitud de paquete de 60 bytes. OpenBSD de 64 bytes. Solaris 7 se queda en el mínimo de 44 bytes, al igual que AIX 4.3, mientras que Microsoft Windows 2000 asciende el tamaño a 48 bytes.

Podemos observar adicionalmente el campo de identificación de IP o *IP ID*. Se utiliza fundamentalmente en fragmentación, y puede ser un gran indicativo a la hora de identificar el sistema subyacente. Por ejemplo, en Linux 2.4 el *IP ID* es totalmente aleatorio, excepto dentro de una misma sesión, durante la cual se incrementa en uno el valor inicial. OpenBSD lo rellena con un número completamente aleatorio. Solaris utiliza una implementación realmente pobre, e incrementa en uno el valor de *IP ID* **permanentemente**, al igual que AIX 4.3 y Windows 2000.

Por último, el *TTL* o "tiempo de vida" juega un papel interesante. El TTL indica el número máximo de saltos (nodos de la red) por los que puede pasar un paquete antes de llegar a su destino. En cada salto se decrementa en uno, y cuando llega a cero el paquete se desecha. El valor inicial de este campo puede ser muy significativo. Mientras que Linux, AIX 4.3 y OpenBSD utilizan un valor inicial de 64, Solaris 7 se desmarca con un valor increíblemente alto de 255 y toda la familia de Windows se queda a medio camino con un *TTL* de 128.

Pero si queremos refinar el análisis y aumentar la fiabilidad de las predicciones debemos tener también en cuenta campos de las cabeceras TCP:

Sorprendentemente, el primer campo que se muestra interesante es el del puerto de origen, pese a que no es determinante a la hora de identificar con exactitud el sistema operativo. Por ejemplo, Red Hat Linux utiliza puertos que varian entre el 1024 y el 4999. Por supuesto eso no significa que no use otros, o que otros sistemas no usen esos puertos. Adicionalmente, si la máquina analizada está tras un encaminador haciendo *NAT*, esta comprobación será inútil.

El valor de la ventana de transmisión es más revelador. Por lo general, los sistemas operativos lo fijan de partida, con lo que es relativamente sencillo identificarlos atendiendo a la ventana inicial observada. Linux utiliza una ventana de 5840 bytes, mientras que Solaris 7 de 8760. Por otra parte, OpenBSD, AIX 4.3 y Windows utilizan la misma ventana inicial de 16384 bytes.

Pero sin duda lo más interesante desde el punto de vista del *fingerprinting* es el campo de opciones TCP. Este campo incluye diversas opciones que se suelen rellenar con *NOP's* para conseguir una longitud total de las opciones múltiplo de 4:

- *MSS* o tamaño máximo de segmento. Por lo general el *MTU* o la unidad máxima de transmisión.
- *Timestamp* o marca de tiempo. Ayudan a medir retardos.
- *Wscale* o escala de la ventana. Sirve tanto para indicar que el receptor está preparado para escalar el tamaño de la ventana, como para, en tal caso, proponer un factor de escala.
- *SackOK* o asentimiento selectivo. Se utiliza para informar al emisor de los paquetes ya recibidos, de forma que éste sólo retransmita aquellos que aún no se han recibido.
- *NOP*. Un sólo byte utilizado, como se ha mencionado antes, para rellenar las opciones hasta que su longitud total sea múltiplo de 4.

Observando las opciones incluidas, el número y posición concreta, podemos predecir con cierta fiabilidad el sistema operativo de la máquina objetivo. Veamos más en detalle cuales son esas opciones según el sistema:

- Linux: `mss 1460, sackOK, timestamp X 0, nop, wscale 0`.
- OpenBSD: `mss 1460, nop, np, sackOK, nop, wscale 0, nop, nop`.
- Solaris 7: `mss 1460`.
- AIX 4.3: `mss 1460`.
- Windows 2000: `mss 1460, nop, nop, sackOK`.

Linux y OpenBSD se diferencian bastante de los demás, al igual que Windows 2000. Desafortunadamente, utilizando sólo esta técnica no podremos diferenciar un Solaris 7 de un AIX 4.3.

Por último, es interesante observar el comportamiento del sistema analizado en situaciones concretas a lo largo del tiempo, no sólo a nivel de un sólo paquete, como por ejemplo el intento de apertura de una conexión [3]. En particular resulta de provecho contabilizar el número de paquetes SYN que se envían reintentando abrir una conexión, y cómo la pila de TCP/IP rellena los campos entre ellos. Veámoslo con ejemplos:

- Red Hat Linux: se reintenta 4 veces (un total de 5 paquetes). Entre el paquete inicial y el segundo transcurre un retardo de 3 segundos, mientras que entre el segundo y el tercero son 6 segundos. El tiempo de retardo entre reintentos se va doblando, de forma que los retardos son en total de 3, 6, 12 y 24 segundos. Por último, y como ya se vio anteriomente, el campo *IP ID* se trata de forma incremental en lugar de aleatoria cuando se trata de establecer una conexión.

- Windows 2000: se reintenta 2 veces (un total de 3 paquetes). El retardo entre paquetes es exactamente idéntico al utilizado en sistemas Linux. El campo *IP ID* es permanentemente incremental, por lo que en este caso es clave el número total de paquetes para discernir entre ambos sistemas.

- OpenBSD: se reintenta 3 veces (un total de 4 paquetes). Los retardos entre paquetes son de 6, 12 y 24 respectivamente. El campo *IP ID* es siempre aleatorio. Esto, sumado a los detalles descritos hasta ahora, hacen muy sencillo identificar este sistema.

- FreeBSD: se reintenta 8 veces (un total de 9 paquetes). La ventana inicial es de 65535, un valor anormalmente alto y que no se utiliza en ningún otro sistema. Por otro lado usa un valor de 1 en la escala de la ventana, caso también excepcional. El resto de opciones TCP se asemejan a las utilizadas por Linux u OpenBSD, salvo en la cantidad de *NOP's*, 3 en total. El identificador de IP se mantiene incremental al igual que en Windows y en Linux. Sin embargo, los tiempos de reenvío no se asemejan a los de otros sistemas, siendo de 3, 3, 3, 3, 4, 6, 12 y 24 segundos en ese orden.

3. Fingerprinting activo

El *fingerprinting* activo se define como aquel en el que el atacante realiza alguna acción que provoque algún tipo de respuesta en la víctima. Esto se traduce en el envío de paquetes destinados a la máquina objetivo, con los que comprobar su comportamiento en situaciones anómalas o no especificadas por los estándares [6]. Las mismas observaciones realizadas en el *fingerprinting* pasivo son interesantes, por lo que cualquier paquete que provoque una respuesta en la que se puedan verificar las condiciones expuestas hasta el momento es un buen comienzo. Adicionalmente, hay multitud de pruebas que se pueden realizar:

- Sonda *FIN*. Se envía un paquete *FIN* a un puerto abierto del objetivo y se espera respuesta. La RFC793 define la ausencia de respuesta como el comportamiento correcto, si bien algunos sistemas como Windows, BSDI, CISCO, MVS, IRIX o HP-UX responden con un *RESET*.

- *Flags* TCP incorrectos. Consiste en el envío de un paquete con *flags* no definidos (64 o 128) en la cabecera de un paquete *SYN*. Las versiones de Linux anteriores a 2.0.35 mantienen el mismo *flag* en sus respuestas. Otros envían un paquete *RESET* de respuesta.

- *ISN* o número de secuencia inicial. Se trata de comprobar el número inicial de secuencia elegido por la víctima en una conexión TCP. FreeBSD, Digital UNIX, IRIX o Solaris utilizan incrementos aleatorios, mientras que Linux o AIX usan números completamente aleatorios. Windows incrementa los números de secuencia con pequeñas cantidades que varían en función del tiempo. Incluso algunas máquinas como *hubs* 3Com o impresoras Apple utilizan siempre el mismo número de secuencia inicial (en este caso concreto, `0xC7001`).

- Bit de no fragmentación. Algunos sistemas operativos como Solaris activan este bit en algunos de sus paquetes para mejorar el rendimiento.
- Asentimientos. Lo sorprendente de esta prueba es que el estándar define estrictamente cómo debe ser el comportamiento de los asentimientos, si bien no todos los sistemas lo siguen de forma adecuada. Se envía un paquete $SYN|PSH|URG$ o $SYN|FIN|URG|PSH$ a un puerto abierto en el destino y se observa el asentimiento recibido. La mayoría de sistemas devolverán el mismo número de secuencia que se les envió, pese a que Windows y algunas impresoras devolverán dicho número incrementado en una unidad.
- Número de mensajes ICMP. La RFC1812 sugiere limitar la tasa de errores enviados, y algunos sistemas hacen uso de dicha sugerencia. En una red sin pérdidas, puede ser interesante provocar respuestas ICMP de error en el sistema objetivo para analizar la cantidad recibidas y el tiempo transcurrido entre ellas. La contrapartida de esta técnica es que las pérdidas la hacen inconsistente y exige un mayor tiempo de análisis.
- Mensajes ICMP citados. Nuevamente en las RFC se especifica que se debe citar parte del paquete que causó el error en una respuesta ICMP. Por ejemplo, en errores de puerto desconocido, casi todas las implementaciones envían la cabecera IP y 8 bytes de vuelta. Sin embargo, Solaris envía ligeramente más información y Linux aún más. Esto permite reconocer ambos sistemas incluso si no tienen puertos abiertos.
- Integridad de mensajes ICMP. Algunos sistemas modifican las cabeceras que citan en sus mensajes de error. Algunos BSD cambian el valor del campo *IP ID* recibido. Otros como AIX o BSDI envían un campo de longitud total con 20 bytes de más. Incluso algunas implementaciones como AIX o FreeBSD, dada la inconsistencia del paquete citado, envían una suma de comprobación o *checksum* nula (0x00).
- Manejo de la fragmentación. Consistente en probar cómo el sistema destino maneja la fragmentación cuando los fragmentos se solapan. Algunos sobreescriben con el último paquete recibido, mientras que otros mantienen los contenidos existentes. Se trata de una técnica difícil de implementar en algunos sistemas como Solaris.
- Inundación de paquetes *SYN*. Una técnica muy agresiva basada en el envío masivo de paquetes *SYN* a la máquina remota. Muchas dejan de aceptar conexiones tras recibir 8 paquetes seguidos. Por lo general los sistemas modernos implementan contramedidas para este tipo de ataques, que además son muy "ruidosos" y pueden tener efectos colaterales no deseados.

4. Implementaciones

La primera prueba de concepto de las técnicas aquí mostradas fue Siphon `http://siphon.sourceforge.net/`. Bastante rudimentaria y simple en cuanto a diseño, apenas incorporaba dos o tres pruebas básicas de *fingerprinting* activo aquí descritas. Parte de su código fuente aún se mantiene en herramientas comerciales. Ni que decir tiene que estas técnicas y la base de datos que utiliza están completamente anticuadas.

p0f `http://lcamtuf.coredump.cx/p0f.shtml` fue la primera herramienta seria de detección de sistema operativo utilizando múltiples pruebas bastante más sofisticadas que las de Siphon. El código de su primera versión se reutilizó en herramientas tan extendidas como Ettercap `http://ettercap.sourceforge.net` o Prelude IDS `http://www.prelude-ids.org/`. Posteriormente una segunda versión mejoró notablemente a la primera, añadiendo entre otras cosas una base de datos más precisa y hasta 16 nuevas pruebas.

QueSO, un polémico software de *fingerprinting* que dió un salto importante respecto al resto de herramientas coetáneas. Sirvió como base e inspiración para la posterior implementación de Nmap `http://www.insecure.org` [6], rivalizando con éste en velocidad y cantidad de sistemas reconocidos, si bien a día de hoy este último no tiene parangón.

Nmap es un escáner de puertos y redes muy com-

pleto y avanzado, que entre otras muchas cosas permite averiguar el sistema operativo ejecutándose en un dispositivo remoto con un grado de acierto sorprendentemente alto. A continuación se muestra un ejemplo del uso de esta herramienta (truncado):

```
# nmap -sT -p 80,3000
-O pantuflo.escet.urjc.es
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20
```

5. Falseando las huellas

Al igual que un administrador de sistemas responsable tomará las medidas necesarias para evitar que los *banners* de sus servicios informen a potenciales atacantes, debe ser consciente de que ello no imposibilita que cualquiera averigüe el sistema operativo que utiliza con sumo detalle, y más en concreto debe estar al día de las técnicas de *fingerprinting*. Para tratar de evitar los ataques descritos en este artículo existen un conjunto de utilidades que permiten falsear los indicios en los que se basan las herramientas más conocidas, como por ejemplo Nmap [7]. A continuación se muestra un ejemplo del efecto que se desea conseguir. Se trata de la salida (truncada) de Nmap contra el propio servidor de dicha herramienta:

```
# nmap -sS -p 80,3000 -O insecure.org
Too many fingerprints match this
host to give specific OS details
```

5.1. Soluciones para Linux

Probablemente la mejor solución que encontremos disponible en sistemas Linux sea IP Personality `http://ippersonality.sourceforge.net/`. Se trata de un módulo para *netfilter* que permite especificar reglas de *iptables* con las que modificar el comportamiento de la pila de red. En dicho comportamiento se incluyen:

- *ISN* o número inicial de secuencia.
- Tamaño inicial de la ventana TCP.
- Opciones de TCP (tipos, valores y su orden dentro del paquete).
- Números identificadores de IP.
- Respuestas a paquetes TCP concretos.
- Respuestas a paquetes UDP concretos.

La mayor ventaja de *IP Personality* es que permite añadir comportamientos mediante archivos de configuración, funcionalidad similar a la que herramientas como QueSO o Nmap incluyen. De esta forma podemos cambiar la "personalidad" de nuestro sistema especificando únicamente la configuración que queremos utilizar, sin importarnos qué parámetros modifica dicha configuración.

Como opción más sencilla encontramos los parches *Stealth* para el núcleo. Disponibles directamente en el núcleo parcheado o en forma de módulo, permiten ignorar paquetes típicos utilizados en pruebas de *fingerprintin*g activo. Una vez parcheado, obtenemos dos nuevas opciones en la configuración:

- *TCP Stack Options*. Indica si queremos utilizar estos parches. Si queremos deshabilitarlos posteriormente podemos hacer uso de los siguientes comandos:

```
echo 0 > /proc/sys/net/ipv4/tcp_ignore_ack
echo 0 > /proc/sys/net/ipv4/tcp_ignore_bogus
echo 0 > /proc/sys/net/ipv4/tcp_ignore_synfin
```

- *Log all dropped packets*. Guarda un registro de todos los paquetes que se han analizado e ignorado. Éstos pueden ser de diferentes tipos:
  - Paquetes con los bits *SYN* y *FIN* activos (`tcp_ignore_synfin`). Esta es la prueba utilizada por el popular QueSO.
  - Paquetes que tengan alguno de los bits reservados activos, o que no tienen ninguno de los siguientes activos: *ACK*, *SYN*, *FIN*, *RST* (`tcp_ignore_bogus`). Permite sortear la segunda prueba utilizada por Nmap.

- Paquetes con los bits *FIN*, *PSH* y *URG* activos (`tcp_ignore_ack`). La prueba número 7 de Nmap.

También en forma de módulo para el núcleo encontramos *Fingerprint Fucker*. Permite simular el comportamiento de otros sistemas operativos, y permite utilizar directamente archivos de configuración de huellas de Nmap. El comportamiento por defecto se corresponde con el de una máquina VAX.

Por último encontramos *IPlog*, un programa que permite llevar un registro sobre la pila TCP/IP del sistema y la detección de algunos escaneos típicos como *XMAS*, *FIN* o *SYN*. Adicionalmente y mediante la opción `-z`, permite evitar las pruebas realizadas por Nmap. El funcionamiento es sencillo. *IPlog* detecta pruebas típicas realizadas por Nmap y actúa en consecuencia enviando respuestas falsas o inesperadas. Si bien esta aproximación es mucho más sencilla de utilizar que otras, no permite la misma flexibilidad, al no facilitar el uso de múltiples "personalidades" en nuestro sistema.

5.2. Soluciones para sistemas BSD

El caso de los sistemas BSD es especialmente interesante. Aparte de la existencia de una versión del ya mencionado *Fingerprint Fucker* para FreeBSD, tanto el propio FreeBSD como OpenBSD incorporan de serie mecanismos de protección contra estos ataques:

- Blackhole: se trata de una opción especial presente en los núcleos BSD para controlar el comportamiento del mismo cuando se reciben paquetes en puertos TCP o UDP cerrados. En el caso de TCP se permite indicar que se devuelva un *RST* (0), se desechen todos los SYN recibidos (1), o se desechen todos los paquetes (2), mientras que en el caso de UDP es posible devolver un mensaje ICMP de *puerto inalcanzable* (0) o simplemente ignorar el paquete (1). Estas opciones se pueden personalizar mediante un par de sencillos comandos:

```
sysctl -w net.inet.tcp.blackhole=[0|1|2]
sysctl -w net.inet.udp.blackhole=[0|1]
```

- FreeBSD *TCP_DROP_SYNFIN*: al igual que la anterior, se trata de una opción especial del núcleo -en este caso de FreeBSD únicamente- que permite ignorar todos los paquetes recibidos con los bits de *SYN* y *FIN* activos al mismo tiempo. Ya que Nmap realiza una prueba con los bits *SYN+FIN+PSH+URG*, como se ha visto antes, esta opción del núcleo conseguirá inhabilitarla.

- *Packet Filter* de OpenBSD: *Packet Filter* es la herramienta más potente en la actualidad para construir encaminadores, cortafuegos y cualquier tipo de dispositivo de red basado en OpenBSD. La propia herramienta hace uso de técnicas de *fingerprinting* para averiguar el sistema operativo de las máquinas remotas cuyos paquetes recibe, proporcionando así al administrador del sistema la posibilidad de hacer cualquier tipo de filtrado en base al sistema operativo. Así mismo, también incluye opciones para eludir o falsear los indicios más tipicos explicados en este documento. En concreto proporciona una serie de directivas de configuración:

  - `no-df`: borra el bit de fragmentación en el paquete actual.
  - `min-ttl`: especifica el tiempo de vida mínimo del paquete actual.
  - `max-mss`: especifica el máximo tamaño de segmento para el paquete actual.
  - `random-id`: sustituye el identificador de IP por valores aleatorios.

5.3. Soluciones generales

Ya que no siempre tendremos la suerte de contar con herramientas como las descritas anteriormente, debemos buscar alternativas que sirvan para ocultar las huellas de cualquier sistema operativo. La mejor idea es anteponer un cortafuegos en el propio encaminador de nuestra red. De esta forma podemos analizar el tráfico y, conociendo las pruebas más comunes realizadas por herramientas de *fingerprinting*, bloquear aquel que se corresponda con dichas pruebas. Cortafuegos hardware como el Checkpoint FW-1

que proporcionan su propio lenguaje de configuración (en este caso concreto *INSPECT*) permiten crear reglas para inspeccionar el tráfico y bloquearlo atendiendo a los requisitos que dispongamos.

6. Conclusiones

En el presente artículo se muestran una serie de técnicas que plantean la base para la implementación de herramientas de *fingerprinting* completas y fiables. Éstas se basan en el análisis del comportamiento de la pila de comunicaciones TCP/IP de los dispositivos remotos, lo cual permite identificar virtualmente cualquier cosa con una dirección IP. Algunas de las herramientas mencionadas alcanzan cotas de acierto realmente altas para una variedad ingente de sistemas operativos.

Por otro lado, el *fingerprinting* se basa en el análisis del comportamiento específico de una implementación, lo que lo convierte en tremendamente variable y sencillo de falsear. En concreto existen múltiples herramientas para inducir comportamientos extraños o análogos a los de otros sistemas operativos en las pruebas más típicas. De este modo un administrador de sistemas precavido puede evitar proporcionar información sensible a posibles atacantes.

Referencias

[1] *Envenenamiento ARP*, Jaime Pérez, 2005.

[2] *Passive OS Fingerprinting: Details and Techniques*, Toby Miller.

[3] *Passive OS Fingerprinting: Details and Techniques (part 2)*, Toby Miller, 2003.

[4] *Silence on the Wire. A Field Guide to Passive Reconnaissance and Indirect Attacks*, Michael Zalewski, 2005, No Starch Press.

[5] *Passive OS Fingerprinting tool*, Michael A. Davis, Kirby Kuehl, Kevin Currie, 2003.

[6] *Remote OS detection via TCP/IP Stack FingerPrinting*, Fyodor, 1998.

[7] *A practical approach for defeating Nmap OS-Fingerprinting*, David Barroso Berrueta, 2003.

[8] *TCP/IP Illustrated Volume I, The Protocols*, W. Richard Stevens, Addison Wesley, 1994.