SISTEMAS DE DETECCIÓN DE INTRUSOS

Antonio Villalón Huerta

Mayo, 2005

Este documento se corresponde casi en su totalidad con el capítulo 18 de *Seguridad en Unix y Redes v2.1*, de Antonio Villalón Huerta.

Índice General

1 Introducción

A pesar de que un enfoque clásico de la seguridad de un sistema informático siempre define como principal defensa del mismo sus controles de acceso (desde una política implantada en un cortafuegos hasta unas listas de control de acceso en un *router* o en el propio sistema de ficheros de una máquina), esta visión es extremadamente simplista si no tenemos en cuenta que en muchos casos esos controles no pueden protegernos ante un ataque ([Lun90]). Por poner un ejemplo sencillo, pensemos en un *firewall* donde hemos implantado una política que deje acceder al puerto 80 de nuestros servidores *web* desde cualquier máquina de Internet; ese cortafuegos sólo comprobará si el puerto destino de una trama es el que hemos decidido para el servicio HTTP, pero seguramente no tendrá en cuenta si ese tráfico representa o no un ataque o una violación de nuestra política de seguridad: por ejemplo, no detendrá a un pirata que trate de acceder al archivo de contraseñas de una máquina aprovechando un *bug* del servidor *web*. Desde un pirata informático externo a nuestra organización a un usuario autorizado que intenta obtener privilegios que no le corresponden en un sistema, nuestro entorno de trabajo no va a estar nunca a salvo de intrusiones.

Llamaremos **intrusión** a un conjunto de acciones que intentan comprometer la integridad, confidencialidad o disponibilidad de un recurso ([HLMS90]); analizando esta definición, podemos darnos cuenta de que una intrusión no tiene por qué consistir en un acceso no autorizado a una máquina: también puede ser una negación de servicio. A los sistemas utilizados para detectar las intrusiones o los intentos de intrusión se les denomina **sistemas de detección de intrusiones** (*Intrusion Detection Systems*, IDS) o, más habitualmente – y aunque no sea la traducción literal – **sistemas de detección de intrusos**; cualquier mecanismo de seguridad con este propósito puede ser considerado un IDS, pero generalmente sólo se aplica esta denominación a los sistemas automáticos (*software* o *hardware*): es decir, aunque un guardia de seguridad que vigila en la puerta de la sala de operaciones pueda considerarse en principio como un sistema de detección de intrusos, como veremos a continuación lo habitual (y lógico) es que a la hora de hablar de IDSes no se contemplen estos casos.

Una de las primeras cosas que deberíamos plantearnos a la hora de hablar de IDSes es si realmente necesitamos uno de ellos en nuestro entorno de trabajo; a fin de cuentas, debemos tener ya un sistema de protección perimetral basado en cortafuegos, y por si nuestro *firewall* fallara, cada sistema habría de estar configurado de una manera correcta, de forma que incluso sin cortafuegos cualquier máquina pudiera seguirse considerando relativamente segura. La respuesta es, sin duda, **sí**; debemos esperar que en cualquier momento alguien consiga romper la seguridad de nuestro entorno informático, y por tanto hemos de ser capaces de detectar ese problema tan pronto como sea posible (incluso antes de que se produzca, cuando el potencial atacante se limite a probar suerte contra nuestras máquinas). Ningún sistema informático puede considerarse completamente seguro, pero incluso aunque nadie consiga violar nuestras políticas de seguridad, los sistemas de detección de intrusos se encargarán de mostrarnos todos los intentos de multitud de piratas para penetrar en nuestro entorno, no dejándonos caer en ninguna falsa sensación de seguridad: si somos conscientes de que a diario hay gente que trata de romper nuestros sistemas, no caeremos en la tentación de pensar que nuestras máquinas están seguras porque nadie sabe de su existencia o porque no son interesantes para un pirata.

Los sistemas de detección de intrusos no son precisamente nuevos: el primer trabajo sobre esta materia ([And80]) data de 1980; no obstante, este es uno de los campos más en auge desde hace ya unos años dentro de la seguridad informática. Y no es extraño: la capacidad para detectar y responder ante los intentos de ataque contra nuestros sistemas es realmente muy interesante. Durante estos veinte años, cientos de investigadores de todo el mundo han desarrollado, con mayor o menor éxito, sistemas de detección de todo tipo, desde simples procesadores de *logs* hasta complejos sistemas distribuidos, especialmente vigentes con el auge de las redes de computadores en los últimos años; una excelente perspectiva histórica de algunos de ellos puede encontrarse en [Lis95] o [Axe98].

2 Clasificación de los IDSes

Generalmente existen dos grandes enfoques a la hora de clasificar a los sistemas de detección de intrusos: o bien en función de **qué** sistemas vigilan, o bien en función de **cómo** lo hacen.

Si elegimos la primera de estas aproximaciones tenemos dos grupos de sistemas de detección de intrusos: los que analizan actividades de una única máquina en busca de posibles ataques, y los que lo hacen de una subred (generalmente, de un mismo dominio de colisión) aunque se emplazen en uno sólo de los *hosts* de la misma. Esta última puntualización es importante: un IDS que detecta actividades sospechosas en una red no tiene porqué (y de hecho en la mayor parte de casos no suele ser así) ubicarse en todas las máquinas de esa red.

- IDSes basados en red.
  Un IDS basado en red monitoriza los paquetes que circulan por nuestra red en busca de elementos que denoten un ataque contra alguno de los sistemas ubicados en ella; el IDS puede situarse en cualquiera de los *hosts* o en un elemento que analice todo el tráfico (como un HUB o un enrutador). Esté donde esté, monitorizará diversas máquinas y no una sola: esta es la principal diferencia con los sistemas de detección de intrusos basados en *host*.

- IDSes basados en máquina.
  Mientras que los sistemas de detección de intrusos basados en red operan bajo todo un dominio de colisión, los basados en máquina realizan su función protegiendo un único sistema; de una forma similar – guardando las distancias, por supuesto – a como actúa un escudo antivirus residente en MS-DOS, el IDS es un proceso que trabaja en *background* (o que despierta periódicamente) buscando patrones que puedan denotar un intento de intrusión y alertando o tomando las medidas oportunas en caso de que uno de estos intentos sea detectado.

Algunos autores ([Gra00]) dividen el segundo grupo, el de los sistemas de detección de intrusos basados en máquina, en tres subcategorías:

- Verificadores de integridad del sistema (SIV).
  Un verificador de integridad no es más que un mecanismo encargado de monitorizar archivos de una máquina en busca de posibles modificaciones no autorizadas, por normal general *backdoors* dejadas por un intruso (por ejemplo, una entrada adicional en el fichero de contraseñas o un `/bin/login` que permite el acceso ante cierto nombre de usuario no registrado). El SIV más conocido es sin duda Tripwire; la importancia de estos mecanismos es tal que en la actualidad algunos sistemas Unix integran 'de serie' verificadores de integridad, como Solaris y su ASET (*Automated Security Enhancement Tools*).

- Monitores de registros (LFM).
  Estos sistemas monitorizan los archivos de *log* generados por los programas – generalmente demonios de red – de una máquina en busca de patrones que puedan indicar un ataque o una intrusión. Un ejemplo de monitor puede ser `swatch`, pero más habituales que él son los pequeños *shellscripts* que casi todos los administradores realizan para comprobar periódicamente sus archivos de *log* en busca de entradas sospechosas (por ejemplo, conexiones rechazadas en varios puertos provenientes de un determinado *host*, intentos de entrada remota como `root...`).

- Sistemas de decepción.
  Los sistemas de decepción o tarros de miel (*honeypots*), como *Deception Toolkit* (DTK), son mecanismos encargados de simular servicios con problemas de seguridad de forma que un pirata piense que realmente el problema se puede aprovechar para acceder a un sistema, cuando realmente se está aprovechando para registrar todas sus actividades. Se trata de un mecanismo útil en muchas ocasiones – por ejemplo, para conseguir 'entretener' al atacante mientras se tracea su conexión – pero que puede resultar peligroso: ¿qué sucede si el propio sistema de decepción tiene un *bug* que desconocemos, y el atacante lo aprovecha para acceder realmente a nuestra máquina?

Realmente esta división queda algo pobre, ya que cada día se avanza más en la construcción de sistemas de detección de intrusos basados en *host* que no podrían englobarse en ninguna de las

subcategorías anteriores.

La segunda gran clasificación de los IDSes se realiza en función de cómo actúan estos sistemas; actualmente existen dos grandes técnicas de detección de intrusos ([Sun96]): las basadas en la detección de anomalías (*anomaly detection*) y las basadas en la detección de usos indebidos del sistema (*misuse detection*). Aunque más tarde hablaremos con mayor profundidad de cada uno de estos modelos, la idea básica de los mismos es la siguiente:

- Detección de anomalías.
  La base del funcionamiento de estos sistemas es suponer que una intrusión se puede ver como una anomalía de nuestro sistema, por lo que si fuéramos capaces de establecer un perfil del comportamiento habitual de los sistemas seríamos capaces de detectar las intrusiones por pura estadística: probablemente una intrusión sería una desviación excesiva de la media de nuestro perfil de comportamiento.

- Detección de usos indebidos.
  El funcionamiento de los IDSes basados en la detección de usos indebidos presupone que podemos establecer patrones para los diferentes ataques conocidos y algunas de sus variaciones; mientras que la detección de anomalías conoce lo normal y detecta lo que no lo es, este esquema se limita a conocer lo anormal para poderlo detectar.

Para ver más claramente la diferencia entre ambos esquemas, imaginemos un sistema de detección basado en monitorizar las máquinas origen desde las que un usuario sospechoso conecta a nuestro sistema: si se tratara de un modelo basado en la detección de anomalías, seguramente mantendría una lista de las dos o tres direcciones más utilizadas por el usuario legítimo, alertando al responsable de seguridad en caso de que el usuario conecte desde otro lugar; por contra, si se tratara de un modelo basado en la detección de usos indebidos, mantendría una lista mucho más amplia que la anterior, pero formada por las direcciones desde las sabemos con una alta probabilidad que ese usuario no va a conectar, de forma que si detectara un acceso desde una de esas máquinas, entonces es cuando el sistema tomaría las acciones oportunas.

En muchos trabajos ([Esc98], [Thu00]...) aparece una tercera clasificación de los IDSes; se trata de la diferenciación entre los sistemas que trabajan periódicamente y los que operan en tiempo real. Nosotros no contemplaremos, aunque la citemos, esta clasificación, ya que es totalmente minoritaria en comparación con las otras dos que hemos comentado. De cualquier forma, la idea es muy simple: un IDS de tiempo real (los denominados *Real–Time Intrusion Detection Systems*) trabaja contínuamente en busca de posibles ataques, mientras que los sistemas que se ejecutan a intervalos (*Vulnerability Scanners*) son analizadores de vulnerabilidades que cualquier administrador ha de ejecutar regularmente (ya sea de forma manual o automática) contra sus sistemas para verificar que no presentan problemas de seguridad.

3 Requisitos de un IDS

Sin importar qué sistemas vigile o su forma de trabajar, cualquier sistema de detección de intrusos ha de cumplir algunas propiedades para poder desarrollar su trabajo correctamente. En primer lugar, y quizás como característica más importante, el IDS ha de ejecutarse contínuamente sin nadie que esté obligado a supervisarlo; independientemente de que al detectar un problema se informe a un operador o se lance una respuesta automática, el funcionamiento habitual no debe implicar interacción con un humano. Podemos fijarnos en que esto parece algo evidente: muy pocas empresas estarían dispuestas a contratar a una o varias personas simplemente para analizar *logs* o controlar los patrones del tráfico de una red. Sin entrar a juzgar la superioridad de los humanos frente a las máquinas (¿puede un algoritmo determinar perfectamente si un uso del sistema está correctamente autorizado?) o viceversa (¿sería capaz una persona de analizar en tiempo real todo el tráfico que llega a un servidor *web* mediano?), hemos de tener presente que los sistemas de detección son mecanismos automatizados que se instalan y configuran de forma que su trabajo habitual sea transparente a los operadores del entorno informático.

Otra propiedad, y también como una característica a tener siempre en cuenta, es la **aceptabilidad** o grado de aceptación del IDS; al igual que sucedía con cualquier modelo de autenticación, los mecanismos de detección de intrusos han de ser aceptables para las personas que trabajan habitualmente en el entorno. Por ejemplo, no ha de introducir una sobrecarga considerable en el sistema (si un IDS ralentiza demasiado una máquina, simplemente no se utilizará) ni generar una cantidad elevada de falsos positivos (detección de intrusiones que realmente no lo son) o de *logs*, ya que entonces llegará un momento en que nadie se preocupe de comprobar las alertas emitidas por el detector. Por supuesto (y esto puede parecer una tontería, pero es algo que se hace más a menudo de lo que podamos imaginar), si para evitar problemas con las intrusiones simplemente apagamos el equipo o lo desconectamos de la red, tenemos un sistema bastante seguro... pero inaceptable.

Una tercera característica a evaluar a la hora de hablar de sistemas de detección de intrusos es la **adaptabilidad** del mismo a cambios en el entorno de trabajo. Como todos sabemos, ningún sistema informático puede considerarse estático: desde la aplicación más pequeña hasta el propio *kernel* de Unix, pasando por supuesto por la forma de trabajar de los usuarios (¿quién nos asegura que ese engorroso procedimiento desde una 'desfasada' línea de órdenes mañana no se realizará desde una aplicación gráfica, que realmente hace el mismo trabajo pero que genera unos patrones completamente diferentes en nuestro sistema?), todo cambia con una periodicidad más o menos elevada. Si nuestros mecanismos de detección de intrusos no son capaces de adaptarse rápidamente a esos cambios, están condenados al fracaso.

Todo IDS debe además presentar cierta tolerancia a fallos o capacidad de respuesta ante situaciones inesperadas; insistiendo en lo que comentábamos antes sobre el carácter altamente dinámico de un entorno informático, algunos – o muchos – de los cambios que se pueden producir en dicho entorno no son graduales sino bruscos, y un IDS ha de ser capaz de responder siempre adecuadamente ante los mismos. Podemos contemplar, por ejemplo, un reinicio inesperado de varias máquinas o un intento de engaño hacia el IDS; esto último es especialmente crítico: sólo hemos de pararnos a pensar que si un atacante consigue modificar el comportamiento del sistema de detección y el propio sistema no se da cuenta de ello, la intrusión nunca será notificada, con los dos graves problemas que eso implica: aparte de la intrusión en sí, la falsa sensación de seguridad que produce un IDS que no genera ninguna alarma es un grave inconveniente de cara a lograr sistemas seguros.

4 IDSes basados en máquina

Como antes hemos comentado, un sistema de detección de intrusos basado en máquina (*host–based IDS*) es un mecanismo que permite detectar ataques o intrusiones contra la máquina sobre la que se ejecuta.

Tradicionalmente, los modelos de detección basados en máquina han consistido por una parte en la utilización de herramientas automáticas de análisis de *logs* generados por diferentes aplicaciones o por el propio *kernel* del sistema operativo, prestando siempre especial atención a los registros relativos a demonios de red, como un servidor *web* o el propio `inetd`, y por otra – quizás no tan habitual como la anterior – en el uso de verificadores de integridad de determinados ficheros vitales para el sistema, como el de contraseñas; no obstante, desde hace unos años un tercer esquema de detección se está implantando con cierta fuerza: se trata de los sistemas de detección, *honeypots* o tarros de miel.

El análisis de *logs* generados por el sistema (entendiendo como tales no sólo a los relativos al núcleo, sino también a aquellos de aplicaciones nativas de cada Unix, como `syslogd`) varía entre diferentes clones de Unix por una sencilla razón: cada uno de ellos guarda la información con un cierto formato, y en determinados ficheros, aunque todos – o casi todos – sean capaces de registrar los mismos datos, que son aquellos que pueden ser indicativos de un ataque. La mayor parte de Unices son capaces de registrar con una granularidad lo suficientemente fina casi todas las actividades que se llevan a cabo en el sistema, en especial aquellas que pueden suponer – aunque sea remotamente – una vulneración de su seguridad; sin embargo, el problema radica en que pocos administradores se preocupan de revisar con un mínimo de atención esos *logs*, por lo que muchos

ataques contra la máquina, tanto externos como internos, y tanto fallidos como exitosos, pasan finalmente desapercibidos. Aquí es donde entran en juego las herramientas automáticas de análisis, como `swatch` o `logcheck`; a grandes rasgos, realizan la misma actividad que podría ejecutar un *shellscript* convenientemente planificado que incluyera entre sus líneas algunos `grep` de registros sospechosos en los archivos de *log*.

¿A qué entradas de estos ficheros debemos estar atentos? Evidentemente, esto depende de cada sistema y de lo que sea 'normal' en él, aunque suelen existir registros que en cualquier máquina denotan una actividad cuanto menos sospechosa. Esto incluye ejecuciones fallidas o exitosas de la orden `su`, peticiones no habituales al servicio SMTP (como `vrfy` o `expn`), conexiones a diferentes puertos rechazadas por *TCP Wrappers*, intentos de acceso remotos como superusuario, etc; si en la propia máquina tenemos instalado un cortafuegos independiente del corporativo, o cualquier otro *software* de seguridad – uno que quizás es especialmente recomendable es *PortSentry* –, también conviene estar atentos a los *logs* generados por los mismos, que habitualmente se registran en los ficheros normales de auditoría del sistema (`syslog`, `messages`...) y que suelen contener información que con una probabilidad elevada denotan un ataque real.

Por otra parte, la verificación de integridad de archivos se puede realizar a diferentes niveles, cada uno de los cuales ofrece un mayor o menor grado de seguridad. Por ejemplo, un administrador puede programar y planificar un sencillo *shellscript* para que se ejecute periódicamente y compruebe el propietario y el tamaño de ciertos ficheros como `/etc/passwd` o `/etc/shadow`; evidentemente, este esquema es extremadamente débil, ya que si un usuario se limita a cambiar en el archivo correspondiente su UID de 100 a 000, este modelo no descubriría el ataque a pesar de su gravedad. Por tanto, parece obvio que se necesita un esquema de detección mucho más robusto, que compruebe aparte de la integridad de la información registrada en el inodo asociado a cada fichero (fecha de última modificación, propietario, grupo propietario...) la integridad de la información contenida en dicho archivo; y esto se consigue muy fácilmente utilizando funciones resumen sobre cada uno de los ficheros a monitorizar, funciones capaces de generar un *hash* único para cada contenido de los archivos. De esta forma, cualquier modificación en su contenido generará un resumen diferente, que al ser comparado con el original dará la voz de alarma; esta es la forma de trabajar de *Tripwire*, el más conocido y utilizado de todos los verificadores de integridad disponibles para entornos Unix.

Sea cual sea nuestro modelo de verificación, en cualquiera de ellos debemos llevar a cabo inicialmente un paso común: generar una base de datos de referencia contra la que posteriormente compararemos la información de cada archivo. Por ejemplo, si nos limitamos a comprobar el tamaño de ciertos ficheros debemos, nada más configurar el sistema, registrar todos los nombres y tamaños de los ficheros que deseemos, para después comparar la información que periódicamente registraremos en nuestra máquina con la que hemos almacenado en dicha base de datos; si existen diferencias, podemos encontrarnos ante un indicio de ataque. Lo mismo sucederá si registramos funciones resumen: debemos generar un *hash* inicial de cada archivo contra el que comparar después la información obtenida en la máquina. Independientemente de los contenidos que deseemos registrar en esa base de datos inicial, siempre hemos de tener presente una cosa: si un pirata consigue modificarla de forma no autorizada, habrá burlado por completo a nuestro sistema de verificación. Así, es **vital** mantener su integridad; incluso es recomendable utilizar medios de sólo lectura, como un CD-ROM, o incluso unidades extraíbles – discos o disquetes – que habitualmente no estarán disponibles en el sistema, y sólo se utilizarán cuando tengamos que comprobar la integridad de los archivos de la máquina.

Por último, aunque su utilización no esté tan extendida como la de los analizadores de *logs* o la de los verificadores de integridad, es necesario hablar, dentro de la categoría de los sistemas de detección de intrusos basados en máquina, de los sistemas de decepción o *honeypots*. Básicamente, estos 'tarros de miel' son sistemas completos o parte de los mismos (aplicaciones, servicios, subentornos...) diseñados para recibir ciertos tipos de ataques; cuando sufren uno, los *honeypots* detectan la actividad hostil y aplican una estrategia de respuesta. Dicha estrategia puede consistir desde un simple correo electrónico al responsable de la seguridad de la máquina hasta un bloqueo automático de la dirección atacante; incluso muchos de los sistemas – la mayoría – son capaces de simular vulnerabilidades conocidas de forma que el atacante piense que ha tenido éxito y prosiga

con su actividad, mientras es monitorizado por el detector de intrusos.

El concepto teórico que está detrás de los tarros de miel es el denominado 'conocimiento negativo': proporcionar deliveradamente a un intruso información falsa – pero que él considerará real – de nuestros sistemas, con diferentes fines: desde poder monitorizar durante más tiempo sus actividades, hasta despistarlo, pasando por supuesto por la simple diversión. Evidentemente, para lograr engañar a un pirata medianamente experimentado la simulación de vulnerabilidades ha de ser muy 'real', dedicando a tal efecto incluso sistemas completos (denominados 'máquinas de sacrificio'), pero con atacantes de nivel medio o bajo dicho engaño es muchísimo más sencillo: en muchos casos basta simular la existencia de un troyano como *BackOrifice* mediante *FakeBO* (`http://cvs.linux.hr/fakebo/`) para que el pirata determine que realmente estamos infectados e intente utilizar ese camino en su ataque contra nuestro sistema.

Algunos sistemas de decepción no simulan vulnerabilidades tal y como habitualmente se suele entender este concepto; dicho de otra forma, su objetivo no es engañar a un atacante durante mucho tiempo, proporcionándole un subentorno en el sistema que aparente de forma muy realista ser algo vulnerable, sino que su 'decepción' es bastante más elemental: se limitan a presentar un aspecto (quizás deberíamos llamarle *interfaz*) que parece vulnerable, pero que cualquier aprendiz de pirata puede descubrir sin problemas que no es más que un engaño. ¿Dónde está entonces la función de estos sistemas, denominados en ocasiones 'detectores de pruebas'? Por norma, su tarea es recopilar información del atacante y del ataque en sí; por ejemplo, ya que antes hemos hablado de *FakeBO*, podemos imaginar un programa que se encargue de escuchar en el puerto 31337 de nuestro sistema – donde lo hace el troyano real – y, cada vez que alguien acceda a él, guardar la hora, la dirección origen, y los datos enviados por el atacante. En realidad, no es una simulación que pueda engañar ni al pirata más novato, pero hemos logrado el objetivo de cualquier sistema de detección de intrusos: registrar el ataque; además, también se puede considerar un *honeypot*, ya que simula un entorno vulnerable, aunque sólo logre engañar a nuestro atacante durante unos segundos. De cualquier forma, es necesario indicar que en algunas publicaciones (como [Gra00]) se diferencia a los sistemas de decepción 'completos' de estos mecanismos de engaño simple, aunque a todos se les englobe dentro del conjunto denominado *honeypots*.

Hemos revisado en este punto las ideas más generales de los sistemas de detección de intrusos basados en *host*; aunque hoy en día los que vamos a describir a continuación, los basados en red, son con diferencia los más utilizados, como veremos más adelante todos son igualmente necesarios si deseamos crear un esquema de detección con la máxima efectividad. Se trata de niveles de protección diferentes pero que tienen un mismo objetivo: alertar de actividades sospechosas y, en algunos casos, proporcionar una respuesta automática a las mismas.

5 IDSes basados en red

Los sistemas de detección de intrusos basados en red (*network–based IDS*) son aquellos capaces de detectar ataques contra diferentes sistemas de una misma red (en concreto, de un mismo dominio de colisión), aunque generalmente se ejecuten en uno solo de los *hosts* de esa red. Para lograr su objetivo, al menos uno de los interfaces de red de esta máquina sensor trabaja en modo promiscuo, capturando y analizando todas las tramas que pasan por él en busca de patrones indicativos de un ataque.

¿Cuáles pueden ser estos 'patrones identificativos de un ataque' a los que estamos haciendo referencia? Casi cualquiera de los diferentes campos de una trama de red TCP/IP (podemos consultar [Ste94], [Com95] o [Tan96] para conocer en profundidad el significado y la utilidad de cada uno de ellos) puede tener un valor que, con mayor o menor probabilidad, represente un ataque real; los casos más habituales incluyen:

- Campos de fragmentación.
  Una cabecera IP contiene dieciseis *bits* reservados a información sobre el nivel de fragmentación del datagrama; de ellos, uno no se utiliza y trece indican el desplazamiento del fragmento que transportan. Los otros dos *bits* indican o bien que el paquete no ha de ser fragmentado

por un *router* intermedio (DF, *Don´t Fragment*) o bien que el paquete ha sido fragmentado y no es el último que se va a recibir (MF, *More Fragments*). Valores incorrectos de parámetros de fragmentación de los datagramas se han venido utilizando típicamente para causar importantes negaciones de servicio a los sistemas y, desde hace también un tiempo incluso para obtener la versión del operativo que se ejecuta en un determinado *host* ([Fyo98]); por ejemplo, ¿qué le sucedería al subsistema de red implantado en el núcleo de una máquina Unix si nunca recibe una trama con el *bit* MF reseteado, indicando que es el último de un paquete? ¿se quedaría permanentemente esperándola? ¿y si recibe uno que en teoría no está fragmentado pero se le indica que no es el último que va a recibir? ¿cómo respondería el núcleo del operativo en este caso? Como vemos, si en nuestras máquinas observamos ciertas combinaciones de *bits* relacionados con la fragmentación realmente tenemos motivos para – cuanto menos – sospechar que alguien trata de atacarnos.

- Dirección origen y destino.
  Las direcciones de la máquina que envía un paquete y la de la que lo va a recibir también son campos interesantes de cara a detectar intrusiones en nuestros sistemas o en nuestra red. No tenemos más que pensar el tráfico proveniente de nuestra DMZ (y que no se trate de la típica respuesta ante una petición como las que se generan al visitar páginas *web*, por poner un ejemplo) que tenga como destino nuestra red protegida: es muy posible que esos paquetes constituyan un intento de violación de nuestra política de seguridad. Otros ejemplos clásicos son las peticiones originadas desde Internet y que tienen como destino máquinas de nuestra organización que no están ofreciendo servicios directos al exterior, como un servidor de bases de datos cuyo acceso está restringido a sistemas de nuestra red.

- Puerto origen y destino.
  Los puertos origen y destino (especialmente este último) son un excelente indicativo de actividades sospechosas en nuestra red. Aparte de los intentos de acceso no autorizado a servicios de nuestros sistemas, pueden detectar actividades que también supondrán *a priori* violaciones de nuestras políticas de seguridad, como la existencia de troyanos, ciertos tipos de barridos de puertos, o la presencia de servidores no autorizados dentro de nuestra red.

- *Flags* TCP.
  Uno de los campos de una cabecera TCP contiene seis *bits* (URG, ACK, PSH, RST, SYN y FIN), cada uno de ellos con una finalidad diferente (por ejemplo, el *bit* SYN es utilizado para establecer una nueva conexión, mientras que FIN hace justo lo contrario: liberarla). Evidentemente el valor de cada uno de estos *bits* será 0 o 1, lo cual de forma aislada no suele decir mucho (ni bueno ni malo) de su emisor; no obstante, ciertas combinaciones de valores suelen ser bastante sospechosas: por ejemplo, una trama con los dos *bits* de los que hemos hablado – SYN y FIN – activados simultáneamente sería indicativa de una conexión que trata de abrirse y cerrarse al mismo tiempo. Para hacernos una idea de la importancia de estos *bits* de control, no conviene olvidar que uno de los problemas de seguridad más conocidos de los últimos años sobre plataformas Windows – de los muchos que han tenido estos entornos – estaba fundamentado básicamente en el manejo de paquetes OOB (*Out Of Band*): tramas con el *bit* URG activado.

- Campo de datos.
  Seguramente, el campo de datos de un paquete que circula por la red es donde más probabilidades tenemos de localizar un ataque contra nuestros sistemas; esto es debido a que con toda probabilidad nuestro cortafuegos corporativo detendrá tramas cuya cabecera sea 'sospechosa' (por ejemplo, aquellas cuyo origen no esté autorizado a alcanzar su destino o con campos incorrectos), pero rara vez un *firewall* se parará a analizar el contenido de los datos transportados en la trama. Por ejemplo, una petición como `'GET ../../../etc/passwd HTTP/1.0'` contra el puerto 80 del servidor *web* de nuestra empresa no se detendrá en el cortafuegos, pero muy probablemente se trata de un intento de intrusión contra nuestros sistemas.

Acabamos de ver **sólo algunos** ejemplos de campos de una trama TCP/IP que, al presentar determinados valores, pueden ser indicativos de un ataque; sin embargo, no todo es tan sencillo como comprobar ciertos parámetros de cada paquete que circula por uno de nuestros segmentos. También es posible y necesario que un detector de intrusos basado en red sea capaz de notificar otros ataques

que no se pueden apreciar en una única trama; uno de estos ataques es la presencia de peticiones que, aunque por sí mismas no sean sospechosas, por su repetición en un intervalo de tiempo más o menos pequeño puedan ser indicativas de un ataque (por ejemplo, barridos de puertos horizontales o verticales). Otros ataques difíciles de detectar analizando tramas de forma independiente son las negaciones de servicio distribuidas (DDoS, *Distributed Denial of Service*), justamente por el gran número de orígenes que el ataque tiene por definición.

Según lo expuesto hasta ahora en este punto, puede parecer que los sistemas de detección de intrusos basados en red funcionan únicamente mediante la detección de patrones; realmente, esto no es así: en principio, un detector de intrusos basado en red puede estar basado en la detección de anomalías, igual que lo puede estar uno basado en máquinas. No obstante, esta aproximación es minoritaria; aunque una intrusión generará probablemente comportamientos anormales (por ejemplo, un tráfico excesivo entre el sistema atacante y el atacado) susceptibles de ser detectados y eliminados, con demasiada frecuencia estos sistemas no detectarán la intrusión hasta que la misma se encuentre en un estado avanzado. Este problema hace que la mayor parte de IDSes basados en red que existen actualmente funcionen siguiendo modelos de detección de usos indebidos.

Para finalizar este punto dedicado a los sistemas de detección de intrusos basados en red es necesario hablar de las *honeynets* ([Spi01]) – literalmente, 'redes de miel' –. Se trata de un concepto muy parecido al de los *honeypots*, de los que ya hemos hablado, pero extendido ahora a redes completas: redes diseñadas para ser comprometidas, formadas por sistemas reales de todo tipo que, una vez penetrados, permiten capturar y analizar las acciones que está realizando el atacante para así poder aprender más sobre aspectos como sus técnicas o sus objetivos. Realmente, aunque la idea general sea común, existen dos grandes diferencias de diseño entre un tarro de miel y una *honeynet*: por un lado, esta última evidentemente es una red completa que alberga diferentes entornos de trabajo, no se trata de una única máquina; por otro, los sistemas dentro de esta red son sistemas reales, en el sentido de que no simulan ninguna vulnerabilidad, sino que ejecutan aplicaciones típicas (bases de datos, sistemas de desarrollo...) similares a las que podemos encontrar en cualquier entorno de trabajo 'normal'. El objetivo de una *honeynet* no es la decepción, sino principalmente conocer los movimientos de un pirata en entornos semireales, de forma que aspectos como sus vulnerabilidades o sus configuraciones incorrectas se puedan extrapolar a muchos de los sistemas que cualquier empresa posee en la actualidad; de esta forma podemos prevenir nuevos ataques exitosos contra entornos reales.

En el funcionamiento de una 'red de miel' existen dos aspectos fundamentales y especialmente críticos, que son los que introducen la gran cantidad trabajo de administración extra que una *honeynet* implica para cualquier organización. Por un lado, tenemos el control del flujo de los datos: es **vital** para nuestra seguridad garantizar que una vez que un sistema dentro de la *honeynet* ha sido penetrado, este no se utilice como plataforma de salto para atacar otras máquinas, ni de nuestra organización ni de cualquier otra; la 'red de miel' ha de permanecer perfectamente controlada, y por supuesto aislada del resto de los segmentos de nuestra organización. En segundo lugar, otro aspecto básico es la captura de datos, la monitorización de las actividades que un atacante lleva a cabo en la *honeynet*. Recordemos que nuestro objetivo principal era conocer los movimientos de la comunidad pirata para poder extrapolarlos a sistemas reales, por lo que también es muy importante para el correcto funcionamiento de una *honeynet* una correcta recogida de datos generados por el atacante: ha de ser capturada toda la información posible de cada acción, de una forma poco agresiva (esto es, sin tener que realizar grandes cambios en cada uno de los sistemas) y por supuesto sin que el atacante se entere. Además (muy importante), estos datos recogidos **nunca** se han de mantener dentro del perímetro de la *honeynet*, ya que si fuera así cualquier pirata podría destruirlos con una probabilidad demasiado elevada.

El concepto de *honeynet* es relativamente nuevo dentro del mundo de la seguridad y, en concreto, de los sistemas de detección de intrusos; a pesar de ello, se trata de una idea muy interesante que presumiblemente va a extenderse de una forma más o menos rápida (no todo lo rápida que nos gustaría, ya que implantar y explotar una *honeynet* no es algo ni trivial, ni mucho menos rápido); cada día más, las herramientas de seguridad no se conforman con detectar problemas conocidos, sino que tratan de anticiparse a nuevas vulnerabilidades que aún no se han publicado pero que pueden estar

– y de hecho están – presentes en multitud de sistemas. Conocer cuanto antes cualquier avance de la comunidad *underground* es algo vital si queremos lograr este objetivo.

Como antes hemos comentado, los sistemas de detección de intrusos basados en red, de los que hemos hablado a lo largo de este punto, son con diferencia los más utilizados actualmente en sistemas en explotación; no obstante, como casi cualquier herramienta relacionada con la seguridad, estos sistemas no son ninguna panacea, y su implantación ha de verse complementada con una correcta configuración de elementos como nuestro cortafuegos corporativo o, por supuesto, los sistemas de detección basados en *host*. Veremos más adelante, en este mismo capítulo, que ambos tipos de IDSes son igualmente necesarios en nuestro entorno de trabajo.

6 Detección de anomalías

Desde que en 1980 James P. Anderson propusiera la detección de anomalías como un método válido para detectar intrusiones en sistemas informáticos ([And80]), la línea de investigación más activa (esto es, la más estudiada, pero no por ello la más extendida en entornos reales) es la denominada *Anomaly Detection IDS*, IDSes basados en la detección de anomalías. La idea es *a priori* muy interesante: estos modelos de detección conocen lo que es 'normal' en nuestra red o nuestras máquinas a lo largo del tiempo, desarrollando y actualizando conjuntos de patrones contra los que comparar los eventos que se producen en los sistemas. Si uno de esos eventos (por ejemplo, una trama procedente de una máquina desconocida) se sale del conjunto de normalidad, automáticamente se cataloga como sospechoso.

Los IDSes basados en detección de anomalías se basan en la premisa de que cualquier ataque o intento de ataque implica un uso anormal de los sistemas ([Ko96], [KS94]...). Pero, ¿cómo puede un sistema conocer lo que es y lo que no es 'normal' en nuestro entorno de trabajo? Para conseguirlo, existen dos grandes aproximaciones ([BB99]): o es el sistema el que es capaz de aprenderlo por sí mismo (basándose por ejemplo en el comportamiento de los usuarios, de sus procesos, del tráfico de nuestra red...) o bien se le especifica al sistema dicho comportamiento mediante un conjunto de reglas. La primera de estas aproximaciones utiliza básicamente métodos estadísticos (medias, varianzas...), aunque también existen modelos en los que se aplican algoritmos de aprendizaje automático; la segunda aproximación consiste en especificar mediante un conjunto de reglas los perfiles de comportamiento habitual basándose en determinados parámetros de los sistemas (con la dificultad añadida de decidir cuáles de esos parámetros que con mayor precisión delimitan los comportamientos intrusivos).

En el primero de los casos (el basado en métodos estadísticos), el detector observa las actividades de los elementos del sistema, activos – sujetos –, pasivos – objetos – o ambos, y genera para cada uno de ellos un perfil que define su comportamiento; dicho perfil es almacenado en el sistema, y se actualiza con determinada frecuencia envejeciendo la información más antigua y priorizando la más fresca. El comportamiento del usuario en un determinado momento se guarda temporalmente en otro perfil, denominado 'perfil actual' (*current profile*), y a intervalos regulares se compara con el almacenado previamente en busca de desviaciones que puedan indicar una anomalía.

En [JV93] se definen diferentes tipos de datos o medidas que pueden ser útiles en la elaboración de estos perfiles:

1. Intensidad de la actividad. Reflejan el ratio de progreso de la actividad en el sistema, para lo cual recogen datos a intervalos muy pequeños – típicamente entre un minuto y una hora –. Estas medidas detectan ráfagas de comportamiento (por ejemplo, una excesiva generación de peticiones de entrada/salida en un cierto intervalo) que en espacios de tiempo más amplios no podrían ser detectadas.

2. Numéricas. Se trata de medidas de la actividad cuyo resultado se puede representar en forma de valor numérico, como el número de ficheros leídos por cierto usuario en una sesión o la cantidad de veces que ese usuario se ha equivocado al teclear su contraseña de acceso al sistema.

3. Categóricas. Las medidas categóricas son aquellas cuyo resultado es una categoría individual, y miden la frecuencia relativa o la distribución de una actividad determinada con respecto a otras actividades o categorías; por ejemplo, cual es la relación entre la frecuencia de acceso a un determinado directorio del sistema en comparación con la de acceso a otro. Seguramente la palabra 'categoría' no es la más afortunada (por lo menos, no la más clara), ya que bajo este término se pueden englobar tanto a objetos (por ejemplo, ficheros) como a eventos (por ejemplo, llamadas a la función `crypt()`) del sistema; esta definición genérica puede resultar más sencilla si distinguimos entre categorías globales e individuales: en castellano plano, podemos entender las categorías globales como acciones muy genéricas dentro de un entorno, mientras que las categorías individuales serían la particularización para un elemento determinado del sistema. Así, una categoría global puede ser la formada por el conjunto de accesos remotos a la máquina, mientras que una individual sería la formada por los accesos desde una determinada ubicación física.

4. Distribución de registros de auditoría. Esta medida analiza la distribución de las actividades generadas en un pasado reciente basándose en los *logs* generados por las mismas; dicho análisis se realiza de forma ponderada, teniendo más peso las actividades más recientes, y es comparado con un perfil de actividades 'habituales' previamente almacenado, de forma que permite detectar si en un pasado reciente se han generado eventos inusuales.

La segunda aproximación a la que antes hemos hecho referencia era la consistente en indicar mediante un conjunto de reglas el comportamiento habitual del sistema; suele ser denominada detección de anomalías basada en especificaciones (*specification–based anomaly detection*), y fué propuesta y desarrollada inicialmente por Calvin Cheuk Wang Ko y otros investigadores de la Universidad de California en Davis, durante la segunda mitad de los noventa ([Ko96], [CKL97]...). La idea en la que se sustentan los sistemas de detección de anomalías basados en especificaciones es que se puede describir el comportamiento 'deseable' (entendiendo por 'deseable' el comportamiento 'normal') de cualquier programa cuya seguridad sea crítica; esta descripción se realiza en base a una especificación de seguridad mediante gramáticas, y se considera una violación de la seguridad – al menos en principio – a las ejecuciones de dichos programas que violen su respectiva especificación.

Para ver más claramente el concepto de la detección de anomalías basada en especificaciones, podemos pensar en la ejecución de un programa que se puede considerar crítico, por ser privilegiado: `/bin/passwd`. Si conseguimos diferenciar las diferentes ejecuciones de esta orden que se pueden considerar 'habituales' (por ejemplo, cuando un usuario cambia su contraseña sin problemas, cuando se equivoca, cuando el sistema no le deja cambiarla por los motivos típicos – es débil, hace poco que la cambió...–), podríamos especificar formalmente cada una de estas secuencias de operación. De esta forma, cada vez que un usuario invoque a `/bin/passwd`, el sistema de detección monitorizará las operaciones que esa llamada genere, y considerará una intrusión a cualquiera que no sea 'habitual'.

La idea de los sistemas de detección de intrusos basados en la detección de anomalías es realmente atractiva; no obstante, existen numerosos problemas a los que estos mecanismos tratan de hacer frente. En primer lugar podemos pararnos a pensar en las dificultades que existen a la hora de 'aprender' o simplemente especificar lo habitual: si alguien piensa que por ejemplo obtener un patrón de tráfico 'normal' en una red es fácil, se equivoca; quizás establecer un conjunto de procesos habituales en una única máquina resulte menos complicado, pero tampoco se trata de una tarea trivial. Además, conforme aumentan las dimensiones de los sistemas (redes con un gran número de máquinas interconectadas, equipos con miles de usuarios...) estos se hacen cada vez más aleatorios e impredecibles.

Otro gran problema de los sistemas basados en detección de anomalías radica en la política de aprendizaje que éstos sigan ([Ran98]); si se trata de esquemas donde el aprendizaje es rápido, un intruso puede generar eventos para conseguir un modelo distorsionado de lo 'normal' antes de que el responsable – humano – de los sistemas se percate de ello, de forma que el IDS no llegue a detectar un ataque porque lo considera algo 'habitual'. Si por el contrario el aprendizaje es lento, el IDS considerará cualquier evento que se aleje mínimamente de sus patrones como algo anómalo, generando un gran número de falsos positivos (falsas alarmas), que a la larga harán que los responsables de

los sistemas ignoren cualquier información proveniente del IDS, con los evidentes riesgos que esto implica.

7 Detección de usos indebidos

Dentro de la clasificación de los sistemas de detección de intrusos en base a su forma de actuar, la segunda gran familia de modelos es la formada por los basados en la detección de usos indebidos. Este esquema se basa en especificar de una forma más o menos formal las potenciales intrusiones que amenazan a un sistema y simplemente esperar a que alguna de ellas ocurra; para conseguirlo existen cuatro grandes aproximaciones ([Ko96]): los sistemas expertos, los análisis de transición entre estados, las reglas de comparación y emparejamiento de patrones (*pattern matching*) y la detección basada en modelos.

Los primeros sistemas de detección de usos indebidos, como NIDES ([L+92]), se basaban en los sistemas expertos para realizar su trabajo; en ellos las intrusiones se codifican como reglas de la base de conocimiento del sistema experto, de la forma genérica *if–then* (*if* CONDICIÓN *then* ACCIÓN). Cada una de estas reglas puede detectar eventos únicos o secuencias de eventos que denotan una potencial intrusión, y se basan en el análisis – generalmente en tiempo real – de los registros de auditoría proporcionados por cualquier sistema Unix: esta es una de las principales ventajas de los sistemas expertos, el hecho de que el mecanismo de registro dentro de Unix venga proporcionado 'de serie', ya que de esta forma el sistema de detección trabaja siempre por encima del espacio del sistema operativo, algo que facilita enormemente su integración dentro de diferentes clones de Unix.

Podemos pensar en un caso real que nos ayude a comprender el funcionamiento de los sistemas expertos a la hora de detectar intrusiones; el típico ejemplo es la detección de un mismo usuario conectado simultáneamente desde dos direcciones diferentes. Cada vez que un usuario se autentica correctamente en el sistema, cualquier Unix genera una línea de registro que se guarda en el fichero de *log* correspondiente; así, al conectar a un servidor Linux Slackware vía SSH, se registra el evento de esta forma:

```
May 27 03:08:27 rosita sshd[5562]: User toni, coming from anita, authenticated.
```

Al leer este registro, un sistema experto comprobaría si el usuario `toni` ya tiene una sesión abierta desde una máquina diferente de `anita`; si esta condición se cumple (recordemos la forma genérica de las reglas del sistema experto, *if–then*) se realizaría la acción definida en la regla correspondiente, por norma generar una alarma dirigida al responsable de seguridad del sistema.

La segunda implementación de los sistemas de detección de usos indebidos era la basada en los análisis de transición entre estados ([PK92]); bajo este esquema, una intrusión se puede contemplar como una secuencia de eventos que conducen al atacante desde un conjunto de estados inicial a un estado determinado, representando este último una violación consumada de nuestra seguridad. Cada uno de esos estados no es más que una imagen de diferentes parámetros del sistema en un momento determinado, siendo el estado inicial el inmediatamente posterior al inicio de la intrusión, y el último de ellos el resultante de la completitud del ataque; la idea es que si conseguimos identificar los estados intermedios entre ambos, seremos capaces de detener la intrusión antes de que se haga efectiva.

Sin duda el sistema de detección basado en el análisis de transición entre estados más conocido es USTAT ([Ilg92]), basado en STAT ([Por92]). Este modelo fué desarrollado inicialmente sobre SunOS 4.1.1 (en la actualidad está portado a Solaris 2), y utiliza los registros de auditoría generados por el *C2 Basic Security Module* de este operativo. En USTAT, estos registros del C2-BSM son transformados a otros de la forma <S, A, O>, representando cada uno de ellos un evento de la forma *'el sujeto S realiza la acción A sobre el objeto O'*; a su vez, cada elemento de la terna anterior está formado por diferentes campos que permiten identificar unívocamente el evento representado. El sistema de detección utiliza además una base de datos – realmente, se trata de simples ficheros planos – formada principalmente por dos tablas, una donde se almacenan las descripciones de los diferentes estados (SDT, *State Description Table*) y otra en la que se almacenan las transiciones

entre estados que denotan un potencial ataque (SAT, *Signature Action Table*). Cuando USTAT registre una sucesión determinada de eventos que representen un ataque entrará en juego el motor de decisiones, que emprenderá la acción que se le haya especificado (desde un simple mensaje en consola informando de la situación hasta acciones de respuesta automática capaces de interferir en tiempo real con la intrusión).

La tercera implementación que habíamos comentado era la basada en el uso de reglas de comparación y emparejamiento de patrones o *pattern matching* ([SG91], [KS94]); en ella, el detector se basa en la premisa de que el sistema llega a un estado comprometido cuando recibe como entrada el patrón de la intrusión, sin importar el estado en que se encuentre en ese momento. Dicho de otra forma, simplemente especificando patrones que denoten intentos de intrusión el sistema puede ser capaz de detectar los ataques que sufre, sin importar el estado inicial en que esté cuando se produzca dicha detección, lo cual suele representar una ventaja con respecto a otros modelos de los que hemos comentado.

Actualmente muchos de los sistemas de detección de intrusos más conocidos (por poner un ejemplo, podemos citar a SNORT o *RealSecure*) están basados en el *pattern matching*. Utilizando una base de datos de patrones que denotan ataques, estos programas se dedican a examinar todo el tráfico que ven en su segmento de red y a comparar ciertas propiedades de cada trama observada con las registradas en su base de datos como potenciales ataques; si alguna de las tramas empareja con un patrón sospechoso, automáticamente se genera una alarma en el registro del sistema. En el punto 8.2 hablaremos con más detalle del funcionamiento de SNORT, uno de los sistemas de detección de intrusos basados en red más utilizado en entornos con requisitos de seguridad media.

Por último, tenemos que hablar de los sistemas de detección de intrusos basados en modelos ([GL91]); se trata de una aproximación conceptualmente muy similar a la basada en la transición entre estados, en el sentido que contempla los ataques como un conjunto de estados y objetivos, pero ahora se representa a los mismos como escenarios en lugar de hacerlo como transiciones entre estados. En este caso se combina la detección de usos indebidos con una deducción o un razonamiento que concluye la existencia o inexistencia de una intrusión; para ello, el sistema utiliza una base de datos de escenarios de ataques, cada uno de los cuales está formado por una secuencia de eventos que conforman el ataque. En cada momento existe un subconjunto de esos escenarios, denominado *de escenarios activos*, que representa los ataques que se pueden estar presentando en el entorno; un proceso denominado *anticipador* analiza los registros de auditoría generados por el sistema y obtiene los eventos a verificar en dichos registros para determinar si la intrusión se está o no produciendo (realmente, al ser esos registros dependientes de cada sistema Unix, el anticipador pasa su información a otro proceso denominado *planner*, que los traduce al formato de auditoría utilizado en cada sistema). El anticipador también actualiza constantemente el conjunto de escenarios activos, de manera que este estará siempre formado por los escenarios que representan ataques posibles en un determinado momento y no por la base de datos completa.

Hasta hace poco tiempo no existían sistemas de detección de intrusos basados en modelos funcionando en sistemas reales ([Kum95]), por lo que era difícil determinar aspectos como su eficiencia a la hora de detectar ataques. En 1998 se presentó NSTAT ([Kem98]), una extensión de USTAT (del cual hemos hablado antes) a entornos distribuidos y redes de computadores, y en el que se combina el análisis de transición entre estados con la detección de intrusos basada en modelos. A pesar de todo, este modelo es el menos utilizado a la hora de detectar ataques y efectuar respuestas automáticas ante los mismos, especialmente si lo comparamos con los basados en la comparación y emparejamiento de patrones.

Los IDSes basados en la detección de usos indebidos son en principio más robustos que los basados en la detección de anomalías: al conocer la forma de los ataques, es teóricamente extraño que generen falsos positivos (a no ser que se trate de un evento autorizado pero muy similar al patrón de un ataque); es necesario recalcar el matiz 'teóricamente', porque como veremos más adelante, la generación de falsos positivos es un problema a la hora de implantar cualquier sistema de detección. No obstante, en este mismo hecho radica su debilidad: sólo son capaces de detectar lo que conocen, de forma que si alguien nos lanza un ataque desconocido para el IDS éste no nos notificará ningún

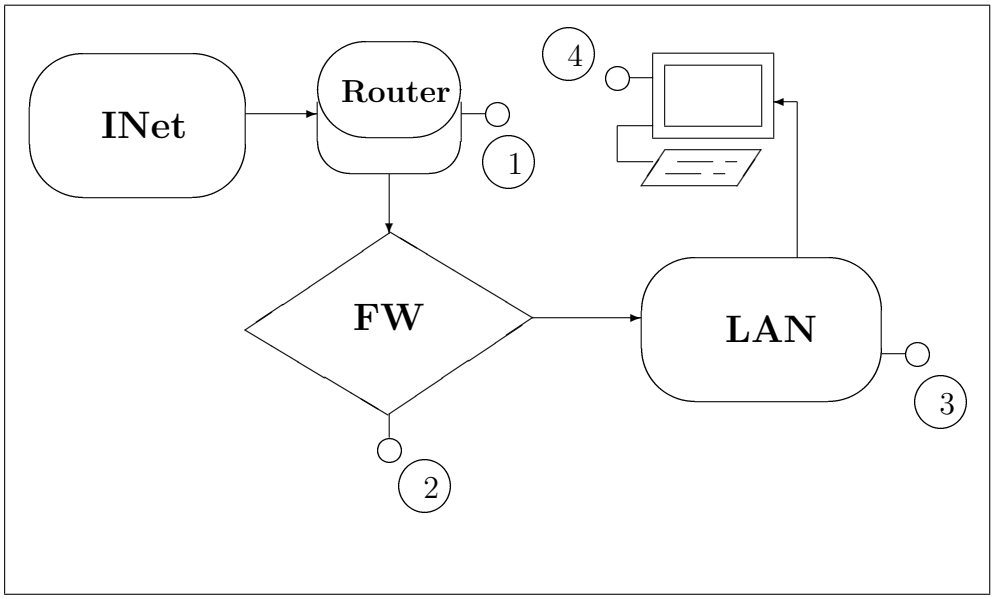

Figura 1: Puntos clásicos de defensa entre un atacante y un objetivo.

problema; como ya dijimos, es algo similar a los programas antivirus, y de igual manera que cada cierto tiempo es conveniente (en MS-DOS y derivados) actualizar la versión del antivirus usado, también es conveniente mantener al día la base de datos de los IDSes basados en detección de usos indebidos. Aún así, seremos vulnerables a nuevos ataques.

Otro grave problema de los IDSes basados en la detección de usos indebidos es la incapacidad para detectar patrones de ataque convenientemente camuflados. Volviendo al ejemplo de los antivirus, pensemos en un antivirus que base su funcionamiento en la búsqueda de cadenas virales: lo que básicamente hará ese programa será buscar cadenas de código hexadecimal pertenecientes a determinados virus en cada uno de los archivos a analizar, de forma que si encuentra alguna de esas cadenas el *software* asumirá que el fichero está contaminado. Y de la misma forma que un virus puede ocultar su presencia simplemente cifrando esas cadenas (por ejemplo de forma semialeatoria utilizando eventos del sistema, como el reloj), un atacante puede evitar al sistema de detección de intrusos sin más que insertar espacios en blanco o rotaciones de *bits* en ciertos patrones del ataque; aunque algunos IDSes son capaces de identificar estas transformaciones en un patrón, otros muchos no lo hacen.

8 Implantación real de un IDS

Vamos a tratar ahora de definir unas pautas para crear un sistema distribuido de detección de intrusos, capaz de generar respuestas automáticas, alarmas, o simplemente *logs* a distintos niveles de nuestra arquitectura de red, formando lo que se suele denominar un modelo de seguridad de círculos concéntricos ([ISV95]). Para ello, imaginemos un pirata externo a nuestra organización que intenta atacar una determinada máquina, y pensemos en el primer punto en el que podemos detectar dicho ataque y actuar sobre él: ahí es donde deberemos implantar el primer sensor, ya que se trata de la primera barrera que estamos interponiendo entre el atacante y su objetivo; ese primer punto no es otro que nuestro *router* de salida a Internet, el marcado como '1' en la figura 1 (pido perdón por mi estilo 'artístico'). No podemos entrar aquí a tratar detalles sobre las capacidades de detección de intrusos de productos como los *routers* Cisco y su IOS, o de otros elementos fácilmente integrables con esta electrónica de red, como NetRanger (también de Cisco), ya que se trata de sistemas que poco tienen que ver con Unix, y que en muchos casos no controlamos nosotros directamente sino una tercera organización (por ejemplo, Telefónica), a pesar de que tengan incluso una dirección IP perteneciente a nuestra red. En las páginas *web* de los distintos fabricantes se puede encontrar

información muy útil sobre sus productos orientados a la detección y respuesta ante ataques.

8.1 IDS en el cortafuegos

Volviendo a la figura 1, el segundo punto que separará al atacante de su objetivo (es decir, nos protegerá de él) será nuestro cortafuegos corporativo. Este elemento, sobre el que probablemente ya tendremos pleno control, estará formado por uno o varios sistemas Unix con un *software* de filtrado de paquetes ejecutándose sobre ellos, y es aquí donde vamos a implantar el primer esquema de detección de intrusos y respuesta automática ante ataques (esta respuesta será habitualmente el bloqueo de la dirección atacante en el propio *firewall*).

Para decidir qué tipos de ataques debemos detectar y bloquear en nuestro cortafuegos debemos pararnos a pensar con qué información trabaja habitualmente este sistema; cualquier *firewall* lo hará al menos con los cinco elementos que definen una conexión bajo la pila TCP/IP: dirección origen, dirección destino, puerto origen, puerto destino y protocolo. De estos cinco, quizás los dos menos importantes (de cara a detectar ataques) son quizás el protocolo utilizado y el puerto origen de la conexión; por tanto, son los otros tres elementos los que nos ayudarán en la constitución de nuestro IDS y los que nos facilitarán el poder lanzar una respuesta automática contra el atacante.

Conociendo las direcciones origen y destino y el puerto destino de una conexión ya podemos detectar cierto tipo de ataques; quizás el ejemplo más habitual son los escaneos de puertos, tanto horizontales como verticales, que se lanzan contra nuestros sistemas. La técnica de detección de estos ataques se define perfectamente en [Nor99]: está basada por el momento en comprobar X eventos de interés dentro de una ventana de tiempo Y. Así, podemos analizar en nuestro cortafuegos cuándo una misma dirección origen accede a un determinado puerto de varios destinos en menos de un cierto tiempo umbral (escaneo horizontal) o cuando accede a diferentes puertos bien conocidos de un mismo sistema también en menos de ese tiempo umbral (escaneo vertical). ¿Por qué el hecho de fijarnos sólo en puertos bien conocidos en este último caso? Muy sencillo: muchas aplicaciones abren muchos puertos destino, generalmente altos (por encima del 1024), en una única sesión de funcionamiento, por lo que esa sesión sería identificada por el cortafuegos como un escaneo vertical cuando realmente no lo es (y si además en nuestro esquema de respuesta automática decidimo bloquear la IP 'atacante', causaríamos una grave negación de servicio contra usuarios legítimos de nuestros sistemas).

Una técnica alternativa que con frecuencia suele ser utilizada con bastante efectividad para detectar escaneos verticales consiste en vigilar del acceso a determinados puertos de los sistemas protegidos por el *firewall*, acceso que con toda probabilidad representará un intento de violación de nuestras políticas de seguridad. No nos engañemos: si alguien trata de acceder desde fuera del segmento protegido a puertos como `echo` (7/TCP,UDP), `systat` (11/TCP), `netstat` (15/TCP), `tcpmux` (1/`tcp`) o el desfasado `uucp` (540/TCP), lo más probable es que se trate de una persona que no lleva muy buena intención con respecto a nuestras máquinas. Seguramente estará lanzando un escaneo vertical contra nosotros, aunque a veces también se puede tratar de un simple curioso que trata de comprobar nuestro grado de seguridad para lanzar un ataque posterior: evidentemente, si alguien tiene abierto un puerto de los citados anteriormente, denota una escasa preocupacion por su seguridad, por lo que casi con toda certeza se puede intuir que tendrá agujeros importantes en alguna de sus máquinas.

Otro tipo de ataques que también son fácilmente detectables vigilando el acceso a determinados puertos de nuestros sistemas protegidos son aquellos que detectan la presencia – o la comprobación de la presencia – de diferentes troyanos como NetBus o BackOrifice: si en el *firewall* se detecta tráfico dirigido a puertos como 12345, 12346 o 20034 (TCP) o como 31337 (UDP), sin duda se trata de un atacante que está tratando de aprovechar estos troyanos; en muchos casos – la mayoría – se tratará de escaneos horizontales en busca de máquinas contaminadas a lo largo de toda o gran parte de nuestra clase C. En la tabla 1 se muestran algunos de los puertos a los que conviene estar atentos a la hora de diseñar una política de detección de intrusos en nuestro cortafuegos; por supuesto, existen muchos más que pueden ser considerados 'sospechosos', pero en cualquier caso siempre conviene ser muy precavido con su monitorización ya que algunos de ellos pueden ser usados por usuarios

Servicio	Puerto	Protocolo	Ataque
`ttymux`	1	TCP	Escaneo horizontal
`echo`	7	TCP/UDP	Escaneo horizontal
`systat`	7	TCP	Escaneo horizontal
`daytime`	13	TCP/UDP	Escaneo horizontal
`netstat`	15	TCP	Escaneo horizontal
`finger`	79	TCP	Escaneo horizontal/vertical
`who`	513	UDP	Escaneo horizontal
`uucp`	540	TCP	Escaneo horizontal/vertical
NetBus	12345	TCP	Troyano
NetBus	12346	TCP	Troyano
NetBus	20034	TCP	Troyano
BackOrifice	31337	UDP	Troyano
Hack´a´Tack	31789	UDP	Troyano
Hack´a´Tack	31790	UDP	Troyano

Tabla 1: Algunos puertos a monitorizar en un *firewall*

lícitos a los que causaríamos una grave negación de servicio si, por ejemplo, les bloqueáramos el acceso a nuestra red a causa de un falso positivo.

Todos sabemos que el cortafuegos es algo vital para proteger a nuestros sistemas, pero lamentablemente es un elemento muy limitado a la hora de detectar ataques. Por ejemplo, imaginemos la siguiente situación: un atacante decide comprobar si nuestro servidor *web* corporativo tiene algún tipo de vulnerabilidad que le pueda ayudar en un ataque contra la máquina; algo muy común hoy en día, ya que quizás uno de los mayores daños que puede sufrir la imagen de una empresa – especialmente si está relacionada con las nuevas tecnologías – es una modificación de su página *web* principal. Es muy probable que ese pirata lanzara en primer lugar un escaneo de puertos vertical contra el servidor, para comprobar si aparte del servicio HTTP se está ofreciendo algún otro; si todo es correcto, el puerto de *web* será el único abierto en el cortafuegos corporativo, cortafuegos que además detectará el ataque contra la máquina y bloqueará, al menos temporalmente, cualquier acceso de la dirección atacante. Un punto a nuestro favor, pero el pirata no tiene más que colgar el módem y volver a llamar a su proveedor para conseguir otra IP, con lo cual obtiene de nuevo acceso a nuestro servicio HTTP y además ya sabe que el único puerto abierto en la máquina es ese.

Ahora ese atacante no necesita ningún tipo de escaneo de puertos adicional; puede seguir varios caminos para atacarnos, pero sin duda el más lógico y fácil es tratar de localizar vulnerabilidades en el servidor *web* de nuestra organización. Para ello puede lanzar un escaneador de vulnerabilidades en servidores *web*[1] contra la máquina, escaneador que no generará ninguna alerta en el cortafuegos; al fin y al cabo, lo único que hacen estos programas es lanzar peticiones al puerto 80 de nuestro servidor *web*, algo que el *firewall* no contempla como sospechoso: para él, no hay diferencia entre un analizador de CGIs y peticiones normales a nuestras páginas.

Parece por tanto evidente que nuestra primera barrera de detección de intrusos es realmente útil, pero también insuficiente frente a determinados ataques; entonces entra en juego el segundo nivel de nuestro sistema de detección de intrusos, el ubicado en el segmento de red – en el dominio de colisión – en el que se encuentra el *host* que estamos tratando de proteger.

8.2 IDS en la red: SNORT

SNORT ([Roe99]) es un *sniffer* capaz de actuar como sistema de detección de intrusos en redes de tráfico moderado; su facilidad de configuración, su adaptabilidad, sus requerimientos mínimos (funciona en diferentes Unices, incluyendo un simple PC con Linux, Solaris o cualquier BSD gratuito),

[1] Generalmente se les denomina *CGI Scanners*, aunque no sólo comprueban la presencia de CGIs vulnerables, sino que también detectan errores de configuración, versiones de los demonios, etc.

y sobre todo su precio (se trata de un *software* completamente gratuito que podemos descargar desde su página oficial en INet, `http://www.snort.org/`) lo convierten en una óptima elección en multitud de entornos, frente a otros sistemas como NFR (*Network Flight Recorder*) o ISS RealSecure que, aunque quizás sean más potentes, son también mucho más pesados e infinitamente más caros.

Para instalar un sistema de detección de intrusos basado en SNORT en primer lugar necesitamos evidentemente este programa, que podemos descargar desde su página *web*. Además, para compilarlo correctamente es necesario disponer de las librerías `libpcap`, un interfaz para tratamiento de paquetes de red desde espacio de usuario, y es recomendable también (aunque no obligatorio) instalar `Libnet`, librería para la construcción y el manejo de paquetes de red. Con este *software* correctamente instalado en nuestro sistema, la compilación de SNORT es trivial.

Si volvemos a la clasificación de IDSes que hemos presentado al principio de este capítulo, podemos clasificar a SNORT como un sistema basado en red (se monitoriza todo un dominio de colisión) y que funciona mediante detección de usos indebidos. Estos usos indebidos – o cuanto menos *sospechosos* – se reflejan en una base de datos formada por patrones de ataques; dicha base de datos se puede descargar también desde la propia página *web* de SNORT, donde además se pueden generar bases de patrones 'a medida' de diferentes entornos (por ejemplo, ataques contra servidores *web*, intentos de negaciones de servicio, *exploits*...). El archivo que utilicemos en nuestro entorno será la base para el correcto funcionamiento de nuestro sistema de detección de intrusos.

Una vez hemos compilado e instalado correctamente el programa llega el momento de ponerlo en funcionamiento; y es aquí donde se produce – al menos inicialmente – uno de los errores más graves en la detección de intrusos. Por lógica, uno tiende a pensar que el sensor proporcionará mejores resultados cuantos más patrones de ataques contenga en su base de datos; nada más lejos de la realidad. En primer lugar, es muy probable que no todos los ataques que SNORT es capaz de detectar sean susceptibles de producirse en el segmento de red monitorizado; si situamos el sensor en una zona desmilitarizada donde únicamente ofrecemos servicio de *web*, ¿qué interés tiene tratar de detectar ataques contra DNS? Lo lógico es que las políticas implementadas en nuestro cortafuegos ni siquiera dejen pasar tráfico hacia puertos que no sean los de los servidores *web* pero, incluso en caso de que el potencial ataque se produjera entre máquinas del propio segmento, hemos de evaluar con mucho cuidado si realmente vale la pena sobrecargar la base de datos con patrones que permitan detectar estos ataques. Evidentemente, cuanta más azúcar más dulce, pero si el sensor ha de analizar todo el tráfico, quizás mientras trata de decidir si un paquete entre dos máquinas protegidas se adapta a un patrón estamos dejando pasar tramas provenientes del exterior que realmente representan ataques: hemos de tener presente que el *sniffer* no detendrá el tráfico que no sea capaz de analizar para hacerlo más tarde, sino que simplemente lo dejará pasar. Así, debemos introducir en la base de patrones de ataques los justos para detectar actividades sospechosas contra nuestra red.

En segundo lugar, pero no menos importante, es necesario estudiar los patrones de tráfico que circulan por el segmento donde el sensor escucha para detectar falsos positivos y, o bien reconfigurar la base de datos, o bien eliminar los patrones que generan esas falsas alarmas. Aunque suene algo crudo, si un patrón nos genera un número considerable de falsos positivos, debemos plantearnos su eliminación: simplemente no podremos decidir si se trata de verdaderas o de falsas alarmas. Esto es especialmente crítico si lanzamos respuestas automáticas contra las direcciones 'atacantes' (por ejemplo, detener todo su tráfico en nuestro *firewall*): volviendo al ejemplo de la zona desmilitarizada con servidores *web*, podemos llegar al extremo de detener a simples visitantes de nuestras páginas simplemente porque han generado falsos positivos; aunque en un entorno de alta seguridad quizás vale la pena detener muchas acciones no dañinas con tal de bloquear también algunos ataques (aunque constituiría una negación de servicio en toda regla contra los usuarios que hacen uso legítimo de nuestros sistemas), en un entorno normal de producción esto es impensable. Seguramente será más provechoso detectar y detener estos ataques por otros mecanismos ajenos al sensor.

En resumen, hemos de adaptar a nuestro entorno de trabajo, de una forma muy fina, la base de datos de patrones de posibles ataques. Quizás valga la pena perder tiempo el tiempo que sea necesario en esta parte de la implantación, ya que eso nos ahorrará después muchos análisis de fal-

sas alarmas y, por qué negarlo, algún que otro susto; una vez tengamos todo configurado, podemos utilizar el siguiente *script* para lanzar SNORT de forma automática al arrancar el sistema (Solaris):

```
anita:~# cat /etc/init.d/snort
#!/sbin/sh
#
# Instalacion:
#       # cp <script> /etc/init.d/snort
#       # chmod 744 /etc/init.d/snort
#       # chown root:sys /etc/init.d/snort
#       # ln /etc/init.d/snort /etc/rc2.d/S99snort
#

# Directorio de log
DIRLOG=/var/log/snort
# Fichero de reglas
RULES=/usr/local/security/snort.conf
# Ejecutable
SNORT=/usr/local/security/snort
# Interfaz
IF=hme0

case "$1" in
'start')
        if [ ! -d "$DIRLOG" ]; then
                mkdir -p "$DIRLOG"
        fi
        if [ ! -r "$RULES" ]; then
                echo "No puedo leer el fichero de patrones..."
                exit -1
        fi
        if [ ! -x "$SNORT" ]; then
                echo "No encuentro el ejecutable..."
                exit -1
        fi
        $SNORT -l $DIRLOG -c $RULES -i $IF -N -D
        ;;
'stop')
        if [ ! -r "/var/run/snort_$IF.pid" ]; then
                echo "No puedo obtener el PID..."
                exit -1
        fi
        kill -TERM `cat /var/run/snort_$IF.pid`
        ;;
*)
        echo "Usage: $0 { start | stop }"
        exit 1
esac
exit 0
anita:~#
```

Con el sensor y sus patrones correctamente configurados ya estamos listos para poner en funcionamiento nuestro sistema de detección de intrusos. Seguramente hasta ahora no hemos tenido muchos problemas con el IDS; no obstante, a partir de ahora las cosas se empiezan a complicar un poco, ya que comienza la segunda parte, la del tratamiento de la información que nuestro sensor nos va a proporcionar. Y es que desde este momento el sistema de detección va a empezar a funcionar y a generar *logs* con notificaciones de posibles ataques, o cuanto menos de actividades sospechosas; es hora de decidir cosas como dónde situar al sensor, qué hacer ante la generación de un evento en el

mismo, cómo procesar la información recibida, o simplemente cuándo rotar los *logs* generados.

El último de los problemas planteados realmente tiene fácil solución; ¿cuándo rotar los *logs* que SNORT genera? La respuesta es muy sencilla: depende. Depende de la cantidad de informes generados en nuestro sensor, depende de la frecuencia con la que debamos realizar informes de los ataque sufridos, depende de la implementación elegida para ejecutar respuestas automáticas ante un ataque (si las ejecutamos), etc. En definitiva, la rotación correcta de unos *logs* es algo que se debe estudiar y planificar para cada entorno concreto, no se puede dar un periodo estricto que se aplique siempre porque sería sin duda erróneo. No obstante, una idea que nos puede ayudar en la toma de esta decisión es la siguiente: rotaremos los *logs* cuando los hayamos procesado y extraído de ellos la información que nos pueda interesar para proteger nuestro entorno.

SNORT genera *logs* en el directorio `/var/log/snort/` si no le indicamos lo contrario (podemos hacerlo con la opción `'-l'` del programa). En ese directorio encontraremos un fichero denominado `alert` con las actividades que se vayan registrando, y, si no hubiéramos especificado la opción `'-N'` al arrancar el programa, una serie de subdirectorios cuyos nombres son las direcciones IP de las máquinas de las que se detecta alguna actividad (es el denominado *'packet logging'*). Como nosotros lo que buscamos es básicamente la generación de alarmas, independiente del *packet logging*, no necesitamos generar estos directorios (aunque nada nos impide hacerlo).

El siguiente *shellscript* planificado convenientemente con `crontab` (si lo ejecutamos más de una vez durante el día quizás nos interese afinar la variable `$FECHA`) puede ser utilizado para realizar la rotación del archivo de alarmas generado por SNORT:

```
anita:~# cat /usr/local/security/rotalog
#!/bin/sh
#
# Directorio de log
DIRLOG=/var/log/snort
# Fecha (DD/MM/YY)
FECHA=‘date +%d.%m.%Y‘
# Interfaz
IF=hme0

if [ ! -d "$DIRLOG" ]; then
        mkdir -p "$DIRLOG"
fi
cd $DIRLOG
mv alert alert-$FECHA
touch alert
chmod 600 alert
kill -HUP ‘cat /var/run/snort_$IF.pid‘
compress alert-$FECHA
anita:~#
```

Independientemente de la rotación de *logs* que llevemos a cabo en cada sensor, suele resultar interesante centralizar todos los *logs* generados en un sólo sistema (a veces se le denomina maestro o *master*), aunque sólo sea para realizar estadísticas, seguimientos de máquinas atacantes y atacadas, o simplemente un *'top ten'* de piratas. Para ello podemos establecer relaciones de confianza entre los sensores y ese maestro para que puedan conectarse entre sí sin necesidad de contraseñas y, de forma automática, transferir los *logs* almacenados y rotados. Por supuesto, a estas alturas dicha relación no la estableceremos mediante la definición de máquinas confiables en archivos `.rhosts` o similares, ni con las herramientas `r-`$*$, sino mediante SSH y las claves públicas y privadas de cada máquina. Aparte de una mayor seguridad (no autenticamos a una máquina simplemente por su dirección o nombre, algo fácilmente falseable), siguiendo un mecanismo de este estilo conseguimos que todas las comunicaciones entre sistemas se realicen de forma cifrada, algo que aquí es muy importante: cualquier información relativa a potenciales ataques o respuestas automáticas a los mismos se ha de considerar como confidencial, por lo que sería un grave error echar todo nuestro trabajo a perder

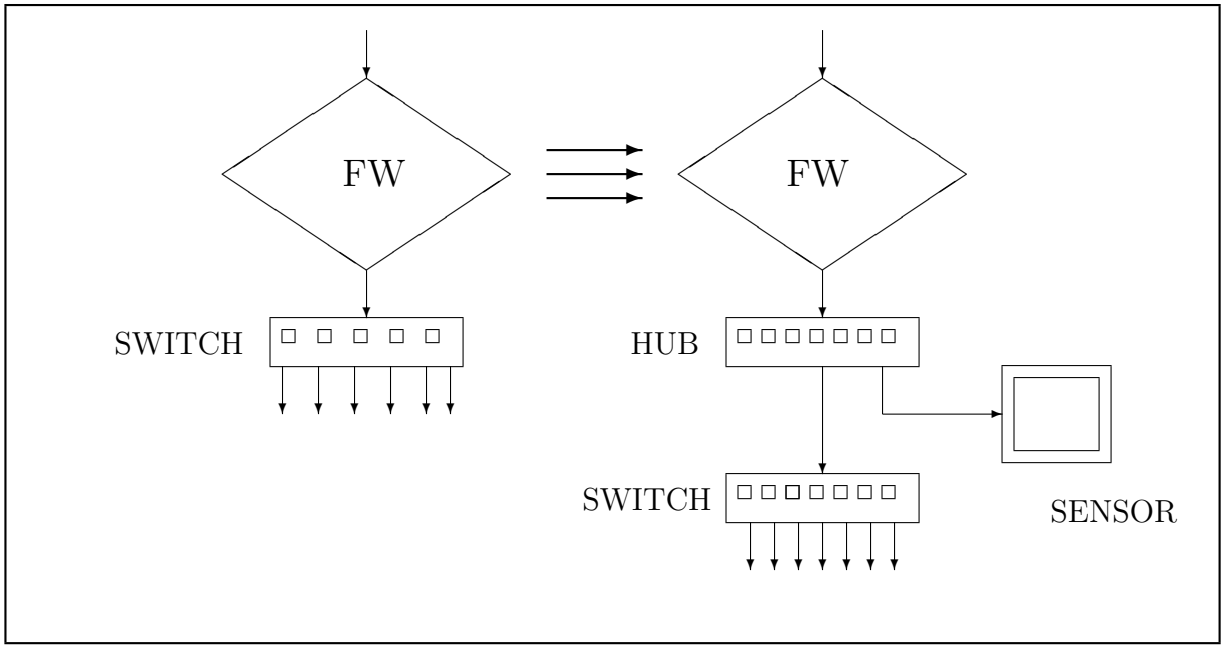

Figura 2: Situación del sensor

simplemente porque alguien sea capaz de esnifar dicho tráfico.

Volviendo a nuestras cuestiones iniciales, también debíamos decidir dónde situar lógicamente al sensor; por ejemplo, una cuestión típica es si debemos emplazarlo detrás o delante del *firewall* que protege a nuestra red. En principio, si dejamos que el sensor analice el tráfico antes de que sea filtrado en el cortafuegos, estaremos en disposición de detectar todos los ataques *reales* que se lanzan contra nuestra red, sin ningún tipo de filtrado que pueda detener las actividades de un pirata; no obstante, probablemente lo que más nos interesará no es detectar todos estos intentos de ataque (aunque nunca está de más permanecer informado en este sentido), sino detectar el tráfico sospechoso que atraviesa nuestro *firewall* y que puede comprometer a nuestros servidores. Por tanto, es recomendable ([Con99]) emplazar el sensor de nuestro sistema de detección de intrusos en la zona protegida; de cualquier forma, los potenciales ataques que no lleguen al mismo quedarán registrados en los *logs* del cortafuegos, e incluso serán neutralizados en el mismo.

Como el sensor ha de analizar todo el tráfico dirigido a las máquinas protegidas, si nos encontramos en un entorno donde dichas máquinas se conecten mediante un concentrador (*hub*) o mediante otras arquitecturas en las que cualquiera de ellas vea (o pueda ver) el tráfico de las demás, no hay muchos problemas de decisión sobre dónde situar al sensor: lo haremos en cualquier parte del segmento. Sin embargo, si nuestros sistemas se conectan con un *switch* la cuestión se complica un poco, ya que en las bocas de este elemento se verá únicamente el tráfico dirigido a las máquinas que estén conectadas a cada una de ellas; en este caso, tenemos varias opciones. Una de ellas puede ser modificar por completo – con todo lo que esto implica – nuestra arquitectura de red para integrar un concentrador por el que pasen los paquetes ya filtrados antes de llegar a las máquinas del *switch*, tal y como se muestra en la figura 2. No obstante, suelen existir alternativas más sencillas y cómodas, como la replicación de puertos que se puede configurar en la mayoría de *switches*; la idea es muy simple: todo el tráfico dirigido a determinada boca del *switch* se monitoriza y se duplica en otra boca. Así, no tenemos más que configurar este *port mirroring* y replicar la boca por la que se dirige el tráfico hacia el segmento de máquinas a monitorizar, enviándolo también a una segunda boca en la que conectaremos nuestro sensor.

Para acabar con los comentarios sobre dónde y cómo situar al sensor de nuestro sistema detector de intrusos, un último apunte: quizás nos conviene recordar que el interfaz por el que se analiza

el tráfico (hablando claro, por el que se esnifan las tramas) no tiene por qué tener dirección IP. Perfectamente podemos tener un interfaz levantado e inicializado pero sin asignarle ninguna dirección. Esto nos puede resultar útil si no nos interesa que en el segmento protegido se detecte una nueva máquina, o simplemente si no queremos que nuestro sensor sea alcanzable de alguna forma por el resto de sistemas de su dominio de colisión. Para nuestra comodidad (por ejemplo, a la hora de centralizar *logs* de diferentes sensores) podemos usar una máquina con dos interfaces, una escuchando todo el tráfico y la otra configurada de forma normal, que será por la que accedamos al sistema.

8.3 IDS en la máquina

Tras leer la sección anterior seguramente habrá quedado claro que un correcto esquema de detección de intrusos basado en red es vital para proteger cualquier sistema; con frecuencia suele ser el punto más importante, que más ataques detecta, y donde se suelen emplazar la mayoría de sistemas de detección que existen instalados en entornos reales hoy en día. No obstante, esta enorme importancia suele degenerar en un error bastante grave: en muchos entornos los responsables de seguridad, a la hora de trabajar con IDSes, se limitan a instalar diversos sensores de detección basados en red en cada segmento a proteger, creyendo que así son capaces de detectar la mayoría de ataques. Y eso suele generar una falsa sensación de seguridad grave, ya que a la hora de lanzar ciertos ataques un pirata puede eludir fácilmente a estos sensores; los sensores de detección en nuestros segmentos de red son importantes, pero no son la panacea. Para comprobarlo, volvamos a nuestro ejemplo anterior, en el que un atacante trata de descubrir vulnerabilidades en nuestros servidores HTTP: si recordamos dónde nos habíamos quedado, el pirata estaba lanzando un escaneador de vulnerabilidades *web* contra el puerto 80 de nuestro servidor corporativo; el esquema de detección implantado en el cortafuegos no percibirá nada extraño, pero el sensor ubicado en el segmento donde se encuentra el servidor sin duda verá patrones que denotan un ataque. Por ejemplo, en el momento en que el escáner compruebe la existencia en el servidor de un CGI denominado `phf`, SNORT generará una alerta similar a esta:

```
[**] IDS128 - CVE-1999-0067 - CGI phf attempt [**]
03/10-03:29:59.834996 192.168.0.3:1032 -> 192.168.0.1:80
TCP TTL:56 TOS:0x0 ID:5040 IpLen:20 DgmLen:58 DF
***AP*** Seq: 0x91FA846  Ack: 0x5AD9A72  Win: 0x7D78  TcpLen: 20
```

Como veremos en el punto siguiente, es posible que al generar este aviso, el propio sensor decida lanzar una respuesta automática contra la dirección atacante (por ejemplo, bloquearla en el cortafuegos corporativo). Pero SNORT trabaja basándose en su base de datos de patrones de ataques; en dicha base de datos habrá una regla como la siguiente:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-CGI phf access";\
flags: A+; content:"/phf";flags: A+; nocase; reference:arachnids,128; \
reference:cve,CVE-1999-0067; )
```

A grandes rasgos, esta regla viene a decir que cuando el sensor detecte una petición al puerto 80 de nuestro servidor *web* en cuyo contenido se encuentre la cadena `'/phf'` levante la alerta correspondiente; de esta forma, cuando el atacante solicite el archivo `/cgi-bin/phf` del servidor, SNORT 'verá' dicha cadena en la petición y generará una alarma. ¿Pero qué sucede si el atacante solicita ese mismo fichero pero utilizando una petición 'ofuscada', formada por los códigos ASCII de cada carácter? Es decir, si en lugar de lanzar una petición como `'GET /cgi-bin/phf'` hace una similar a `'GET %2f%63%67%69%2d%62%69%6e%2f%70%68%66'`. La respuesta es sencilla: la petición anterior se 'decodifica' en el propio servidor *web*, o como mucho en un *proxy* intermedio. Algunos detectores de intrusos, como RealSecure, son en teoría capaces de procesar este tipo de tráfico y analizarlo normalmente en busca de patrones sospechosos, pero otros muchos (SNORT en este caso) no; así, estos últimos no verán en ningún momento la cadena `'/phf'` circulando por la red, y por tanto no generarán ninguna alarma. Parece entonces evidente que es necesario un nivel adicional en nuestro esquema de detección: justamente el compuesto por los sistemas basados en la propia máquina a proteger.

Antes hemos hablado de los tres modelos básicos de IDSes basados en máquina: verificadores

de integridad, analizadores de registros y *honeypots*. Parece claro que un verificador de integridad en nuestro caso no va a resultar muy útil para detectar a ese atacante (esto no significa que no se trate de modelos necesarios y útiles en otras muchas situaciones). Tendremos por tanto que recurrir a analizadores de *logs* o a tarros de miel instalados en la máquina. Si optamos por los primeros, es sencillo construir un *shellscript* que procese los archivos generados por el servidor *web* – en nuestro caso, aunque igualmente sencillo que procesar registros del sistema o de otras aplicaciones – en busca de patrones que puedan denotar un ataque, como el acceso a determinados archivos bajo el `DocumentRoot`. Si analizamos cualquier *CGI Scanner* nos podremos hacer una idea de a qué patrones debemos estar atentos: por ejemplo, intentos de acceso a CGIs como `phf` o `printenv`, a archivos `passwd`, a ficheros fuera del `DocumentRoot`, etc.

Aunque analizar los registros generados por una aplicación en busca de ciertos patrones sospechosos es una tarea trivial, no lo es tanto el integrar ese análisis en un esquema de detección de intrusos. Seguramente procesar la salida de un `'tail -f'` del archivo de *log* correspondiente, enviando un correo electrónico al responsable de seguridad cuando se detecte una entrada sospechosa es fácil, pero por poner un simple ejemplo, ¿qué sucede cuando la máquina se reinicia o el fichero de registros se rota? ¿Es necesario volver a entrar y lanzar de nuevo la orden correspondiente para analizar los *logs*? Seguramente alguien planteará que se puede planificar una tarea para que periódicamente compruebe que se está procesando el archivo, y lance el análisis en caso de que no sea así... ¿pero hasta qué punto es esto cómodo? ¿qué sucede con las entradas duplicadas? ¿y con los registros perdidos que no se llegan a procesar? Evidentemente, no todo es tan sencillo como el simple `'tail -f'` anterior.

El análisis de registros generados por el sistema o por ciertas aplicaciones suele ser una excelente fuente de información de cara a la detección de actividades sospechosas en nuestros entornos, pero no es habitual aplicar dicho análisis en un esquema de respuesta automática ante ataques. Es mucho más común automatizar la revisión de *logs* para que periódicamente (por ejemplo, cada noche) se analicen esos registros y se envíe un mensaje de alerta por correo electrónico a los responsables de seguridad en caso de que algo anormal se detecte; evidentemente no es un modelo que trabaje en tiempo real, pero no por ello deja de ser un esquema útil en la detección de intrusos; el único problema que se presenta a la hora de realizar el análisis suele ser la decisión de qué patrones buscar en los registros, algo que depende por completo del tipo de *log* que estemos analizando.

Aparte de los sistemas de detección basados en el análisis de registros, otro esquema que podemos – y debemos – implantar en cada máquina son los *honeypots* o tarros de miel, mecanismo que nos ha de permitir detectar ciertos ataques aunque el resto de sensores de nuestro modelo falle. Como antes hemos comentado, hay diferentes modelos de tarros de miel, desde los simples detectores de pruebas hasta los complejos sistemas dedicados por completo a esta tarea; para detectar ataques rutinarios estos últimos suelen ser algo excesivo e incluso en ocasiones difícil no sólo desde un punto de vista estrictamente técnico, ya que no siempre podemos dedicar una máquina entera a 'engañar' atacantes, aparte del tiempo necesario para configurarla correctamente, actualizarla, revisar sus registros, etc. Esquemas teóricamente más simples pueden incluso resultar más efectivos en la práctica.

En nuestro caso no vamos a complicarnos mucho la existencia; si recordamos a nuestro atacante, estaba lanzando un escaneo de vulnerabilidades contra nuestro servidor *web*; nuestro IDS basado en red detectará el ataque y en consecuencia dará la voz de alarma y, adicionalmente, ejecutará una respuesta automática contra el pirata, bloqueándolo en el cortafuegos corporativo. Ese atacante ya puede más que sospechar que tenemos un detector de intrusos basado en red que le impide completar su escaneo, ya que en el momento que se detecta tráfico sospechoso bloquea por completo a la dirección origen; por tanto, un paso lógico para él es intentar un análisis de vulnerabilidades 'camuflado', bien mediante una simple codificación de peticiones bien mediante un complejo *stealth scan*. De nuevo, buscará la existencia de CGIs vulnerables, como `/cgi-bin/phf` o `/cgi-bin/printenv`, y en este caso SNORT será incapaz de detectar el ataque. Pero forzosamente a nuestro servidor *web* han de llegar estas peticiones, por lo que ¿qué mejor forma de detectar al pirata que en la propia máquina? No tenemos más que situar en el directorio donde se ubican nuestros CGIs un programa que nos informe si alguien intenta acceder a él, tan simple como el siguiente *shellscript*:

```
anita:~# cat /web/cgi-bin/phf
#!/bin/sh
/bin/echo "Pirata desde "$REMOTE_ADDR""|mailx -s "Ataque PHF" root
/bin/echo "Content-type: text/html"
/bin/echo ""
/bin/echo "<HTML><CENTER>Sonrie a la camara oculta...</CENTER></HTML>"
anita:~#
```

Evidentemente la 'decepción' del sistema anterior en un atacante durará unos pocos segundos, ya que es inmediato para éste detectar que se trata de una simple trampa; si queremos algo más elaborado, tampoco es difícil conseguirlo: podemos simular el comportamiento de diferentes CGIs con vulnerabilidades conocidas de una forma muy sencilla. Por ejemplo, existen diferentes 'imitaciones' de CGIs vulnerables que aparentan un problema de seguridad pero que en realidad se trata de sistemas de decepción más o menos eficaces; en `http://www.eng.auburn.edu/users/rayh/software/phf.html` podemos encontrar una versión muy interesante del CGI `phf`, capaz de simular los ataques más habituales contra el programa original de una forma bastante creíble – aunque no perfecta –. Además en este caso el código en *Perl* es fácilmente modificable, con lo que podemos desde adecuarlo a nuestros sistemas hasta mejorar su simulación; si lo hacemos, es muy posible que mantengamos 'entretenidos' incluso a piratas con conocimientos por encima de la media de atacantes (esto lo digo por experiencia). También podemos construirnos nuestros propios CGIs que aparenten ser vulnerables, en muchos casos simplemente con unos conocimientos básicos de programación en *shell* o en *Perl*; por ejemplo, el código siguiente simula el CGI `printenv`, proporcionando información falsa sobre el sistema que parece útil para un atacante, y avisando por correo electrónico a los responsables de seguridad del sistema cuando alguien accede al programa:

```
anita:/# cat /web/cgi-bin/printenv
#!/usr/bin/perl

$SECUREADDRESS="root";
$mailprog = '/usr/lib/sendmail';

print "Content-type: text/html\n\n";
print "SERVER_SOFTWARE = Apache/1.3.12<br>";
print "GATEWAY_INTERFACE = CGI/1.2<br>";
print "DOCUMENT_ROOT = /usr/local/httpd/htdocs<br>";
print "REMOTE_ADDR = $ENV{'REMOTE_ADDR'}<br>";
print "SERVER_PROTOCOL = $ENV{'SERVER_PROTOCOL'}<br>";
print "SERVER_SIGNATURE = <br><i>Apache/1.3.9 Server at \
       $ENV{'SERVER_NAME'}</i><br><br>";
print "REQUEST_METHOD = GET<br>";
print "QUERY_STRING = $ENV{'QUERY_STRING'}<br>";
print "HTTP_USER_AGENT = $ENV{'HTTP_USER_AGENT'}<br>";
print "PATH = /sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:\
       /usr/local/bin<br>";
print "HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg<br>";
print "HTTP_CONNECTION = Keep-Alive<br>";
print "REMOTE_PORT = $ENV{'REMOTE_PORT'}<br>";
print "SERVER_ADDR = $ENV{'SERVER_ADDR'}<br>";
print "HTTP_ACCEPT_LANGUAGE = en<br>";
print "SCRIPT_NAME = /cgi-bin/printenv<br>";
print "HTTP_ACCEPT_ENCODING = gzip<br>";
print "SCRIPT_FILENAME = /usr/local/httpd/cgi-bin/printenv<br>";
print "SERVER_NAME = $ENV{'SERVER_NAME'}<br>";
print "REQUEST_URI = /cgi-bin/printenv<br>";
print "HTTP_ACCEPT_CHARSET = iso-8859-1, utf-8<br>";
print "SERVER_PORT = $ENV{'SERVER_PORT'}<br>";
print "HTTP_HOST = $ENV{'HTTP_HOST'}<br>";
print "SERVER_ADMIN = webmaster<br>";
```

```
# Enviamos correo
open (MAIL, "|$mailprog $SECUREADDRESS") or die "Can't open $mailprog!\n";
print MAIL "To: $SECUREADDRESS\n";
print MAIL "From: PRINTENV Watcher <$SECUREADDRESS>\n";
print MAIL "Subject: [/CGI-BIN/PRINTENV] $ENV{'REMOTE_HOST'} $action\n\n";
print MAIL "\n";
print MAIL  "------------------------------------------------------\n";
print MAIL "Remote host: $ENV{'REMOTE_ADDR'}\n";
print MAIL "Server: $ENV{'SERVER_NAME'}\n";
print MAIL "Remote IP address: $ENV{'REMOTE_ADDR'}\n";
print MAIL "HTTP Referer: $ENV{'HTTP_REFERER'}\n";
print MAIL "Query String: $ENV{'QUERY_STRING'}\n";
print MAIL "\n------------------------------------------------------\n";
close(MAIL);
exit;
anita:/#
```

8.4 Estrategias de respuesta

Una pregunta importante que nos habíamos realizado con respecto a nuestro esquema de detección de intrusos (en concreto cuando hablábamos de SNORT) era qué hacer con la información obtenida del mismo; como siempre, tenemos varias posibilidades. Por un lado, podemos procesarla manualmente de forma periódica (por ejemplo cada mañana) y en función de los datos que nuestros sensores hayan recogido tomar una determinada acción: bloquear las direcciones atacantes en el *firewall* corporativo, enviar un correo de queja a los responsables de dichas direcciones (por desgracia esto no suele resultar muy útil), realizar informes para nuestros superiores, o simplemente no hacer nada. Dependerá casi por completo de la política de seguridad que sigamos o que tengamos que seguir.

Mucho más interesante que cualquiera de estas respuestas manuales es que el propio sensor sea capaz de generar respuestas automáticas ante lo que él considere un intento de ataque. Si elegimos esta opción, lo más habitual suele ser un bloqueo total de la dirección atacante en nuestro cortafuegos, unida a una notificación de cualquier tipo (SMS, *e-mail*...) a los responsables de seguridad; es siempre recomendable efectuar esta notificación (si no en tiempo real, si al menos registrando las medidas tomadas contra una determinada dirección en un *log*) por motivos evidentes: si bloqueamos completamente el acceso de alguien hacia nuestros sistemas, debemos pararnos a pensar que es posible que se trate de un usuario autorizado que en ningún momento atacaba nuestras máquinas, a pesar de que el sensor que hemos instalado opine lo contrario. No importa lo seguros que estemos de lo que hacemos ni de las veces que hayamos revisado nuestra base de datos de patrones: nunca podemos garantizar por completo que lo que nuestro IDS detecta como un ataque realmente lo sea.

Un correcto esquema de respuesta automática (asumimos que vamos a bloquear la dirección que presuntamente nos ataca) debería contemplar al menos los siguientes puntos:

- ¿Qué probabilidad hay de que el ataque detectado sea realmente un falso positivo?
  No todos los ataques que un sensor detecta tienen la misma probabilidad de serlo realmente: por ejemplo, alguien que busca en nuestros servidores *web* un CGI denominado `phf` (un programa que se encuentra en versiones antiguas – muy antiguas – de Apache, y que presenta graves problemas de seguridad) casi con toda probabilidad trata de atacar nuestros sistemas; en cambio, una alarma generada porque el sensor detecta un paquete ICMP de formato sospechoso no tiene por qué representar (y seguramente no lo hará) un verdadero ataque. Es necesario, o cuanto menos recomendable, asignar un peso específico a cada alarma generada en el sensor, y actuar sólo en el caso de que detectemos un ataque claro: o bien un evento que denote muy claramente un intento de intrusión, o bien varios menos sospechosos, pero cuya cantidad nos indique que no se trata de falsos positivos.

- ¿Debemos contemplar direcciones 'protegidas'?
  Otro punto a tener en cuenta antes de lanzar una respuesta automática contra un potencial

atacante es su origen; si se trata de una dirección externa, seguramente la respuesta será adecuada, pero si se trata de una interna a nuestra red (o equivalentemente, de direcciones de empresas colaboradoras, aunque sean externas) la cosa no está tan clara. Debemos plantearnos que quizás estamos bloqueando el acceso a alguien a quien, por los motivos que sea, no deberíamos habérselo bloqueado. Aunque se trate de cuestiones mas políticas que técnicas, es muy probable que en nuestro IDS debamos tener claro un conjunto de direcciones contra las que no se va actuar; si desde ellas se detecta actividad sospechosa, podemos preparar un mecanismo que alerte a los responsables de seguridad y, bajo la supervisión de un humano, tomar las acciones que consideremos oportunas.

- ¿Hay un límite al número de respuestas por unidad de tiempo?
  Seguramente la respuesta inmediata a esta pregunta sería 'no'; a fin de cuentas, si me atacan diez direcciones diferentes en menos de un minuto, ¿qué hay de malo en actuar contra todas, bloqueándolas? En principio esta sería la forma correcta de actuar, pero debemos pararnos a pensar en lo que es normal y lo que no lo es en nuestro entorno de trabajo. ¿Es realmente habitual que suframos ataques masivos y simultáneos desde diferentes direcciones de Internet? Dependiendo de nuestro entorno, se puede considerar una casualidad que dos o tres máquinas diferentes decidan atacarnos al mismo tiempo; pero sin duda más de este número resulta cuanto menos sospechoso. Un pirata puede simular ataques desde diferentes *hosts* de una forma más o menos sencilla y si nos limitamos a bloquear direcciones (o redes completas), es fácil que nosotros mismos causemos una importante negación de servicio contra potenciales usuarios legítimos (por ejemplo, simples visitantes de nuestras páginas *web* o usuarios de correo), lo cual puede representar un ataque a nuestra seguridad más importante que el que causaría ese mismo pirata si simplemente le permitiéramos escanear nuestras máquinas. Si nuestro sensor lanza demasiadas respuestas automáticas en un periodo de tiempo pequeño, el propio sistema debe encargarse de avisar a una persona que se ocupe de verificar que todo es normal.

Teniendo en cuenta los puntos anteriores (y seguramente otros), debemos diseñar un esquema de respuestas automáticas adecuado. Un pseudoalgoritmo del funcionamiento de nuestro esquema podría ser el siguiente:

```
                                ESTADO: {Detección ataque}
SI umbral de respuestas superado: FIN
SI NO
        Comprobación IP atacante
        SI es IP protegida: FIN
        SI NO
                Ponderación histórica de gravedad
                SI umbral de gravedad superado: ACTUAR
                SI NO
                        REGISTRAR
                FSI
        FSI
FSI
```

Como vemos, al detectar un ataque desde determinada dirección, el modelo comprueba que no se ha superado el número de respuestas máximo por unidad de tiempo (por ejemplo, que no se ha bloqueado ya un número determinado de direcciones en las últimas 24 horas); si este umbral ha sido sobrepasado no actuaremos de ninguna forma automática, principalmente para no causar importantes negaciones de servicio, pero si no lo ha sido, se verifica que la dirección IP contra la que previsiblemente se actuará no corresponde a una de nuestras 'protegidas', por ejemplo a una máquina de la red corporativa: si es así se finaliza. Este 'finalizar' no implica ni mucho menos que el ataque no haya sido detectado y se haya guardado un *log* del mismo, simplemente que para evitar problemas mayores no actuamos en tiempo real contra la máquina: si corresponde al grupo de 'protegidas' es porque tenemos cierta confianza – o cierto control – sobre la misma, por lo que un operador puede preocuparse más tarde de investigar la alerta. Si se trata de una dirección no

controlada ponderamos la gravedad del ataque: un ataque con poca posibilidad de ser un falso positivo tendrá un peso elevado, mientras que uno que pueda ser una falsa alarma tendrá un peso bajo; aún así, **diferentes** ataques de poco peso pueden llegar a sobrepasar nuestro límite si se repiten en un intervalo de tiempo pequeño. Es importante recalcar el 'diferentes', ya que si la alarma que se genera es todo el rato la misma debemos plantearnos algo: ¿es posible que un proceso que se ejecuta periódicamente y que no se trata de un ataque esté todo el rato levantando una falsa misma alarma? La respuesta es **sí**, y evidentemente no debemos bloquearlo sin más; seguramente es más recomendable revisar nuestra base de datos de patrones de ataques para ver por qué se puede estar generando contínuamente dicha alarma.

Por último, si nuestro umbral no se ha superado debemos registrar el ataque, su peso y la hora en que se produjo para poder hacer ponderaciones históricas ante más alarmas generadas desde la misma dirección, y si se ha superado el esquema debe efectuar la respuesta automática: bloquear al atacante en el cortafuegos, enviar un correo electrónico al responsable de seguridad, etc. Debemos pensar que para llegar a este último punto, tenemos que estar bastante seguros de que realmente hemos detectado a un pirata: no podemos permitirnos el efectuar respuestas automáticas ante cualquier patrón que nos parezca sospechoso sin saber realmente – o con una probabilidad alta – que se trata de algo hostil a nuestros sistemas.

Antes de finalizar este punto, es necesario que volver a insistir una vez más en algo que ya hemos comentado: es **muy recomendable** que ante cada respuesta se genere un aviso que pueda ser validado por un administrador de sistemas o por responsables de seguridad, mejor si es en tiempo real; por muy seguros que estemos del correcto funcionamiento de nuestro detector de intrusos, nadie nos garantiza que no nos podamos encontrar ante comportamientos inesperados o indebidos, y con toda certeza es más fácil para una persona que para una máquina darse cuenta de un error y subsanarlo.

8.5 Ampliación del esquema

Las ideas que acabamos de comentar pueden resultar más o menos interesantes, pero presentan varios problemas importantes que es necesario comentar. El primero y más importante es la descentralización del esquema: tenemos implantadas varias aproximaciones a la detección de intrusos, pero hasta ahora no hemos hablado de la relación entre ellas; cada uno de los modelos de detección y/o respuesta de los que hemos tratado puede actuar de forma independiente sin muchos problemas, pero en los entornos actuales esto es cada vez menos habitual. Hoy en día, lo normal es encontrarse arquitecturas de red segmentadas, con sensores en cada segmento tanto a nivel de red como de *host*, así como uno o varios cortafuegos corporativos en los que también se lleva a cabo detección de intrusos y respuesta automática. Está claro que tener elementos independientes no es la aproximación más adecuada, por lo que necesitamos un esquema capaz de unificar los ataques detectados, por ejemplo para correlar eventos y lanzar únicamente una respuesta automática ante un mismo ataque, aunque se detecte simultáneamente en diferentes sensores; sin ser tan ambiciosos, la centralización en una única consola puede ser necesaria para algo tan simple como generar estadísticas mensuales acerca del número de ataques contra nuestro entorno de trabajo: si un mismo ataque se detecta en varios sensores, ¿cómo contabilizarlo? ¿cómo saber si se trata del mismo intruso o de varios? ¿quién de todos decide lanzar una respuesta automática?...

Para tratar de solucionar este importante problema, hace unos años se definió el grupo de trabajo IDWG (*Intrusion Detection Exchange Format Working Group*), englobado dentro del IETF (*Internet Engineering Task Force*); su propósito es obvio: tratar de definir estándares para el intercambio de información entre los diferentes elementos de un sistema de detección de intrusos y respuesta automática, tanto a nivel de formato de datos como de procedimientos de intercambio. Mediante esta aproximación se facilita enormemente tanto la integración de sistemas, ya que de lo único que nos debemos preocupar es que todos los elementos (sensores, consolas, elementos de respuesta...) sean capaces de 'hablar' el estándar, como la escalabilidad: añadir a un entorno de detección distribuido un nuevo IDS, comercial o desarrollo propio, es mucho más sencillo, ya que casi sólo hemos de conseguir que el nuevo elemento utilice los formatos estándar definidos por el IDWG.

Hasta la fecha, el grupo de trabajo ha publicado cuatro borradores que cubren los requisitos (de alto nivel) para las comunicaciones dentro del sistema de detección de intrusos, el modelo de datos para representar la información relevante para los IDSes – incluyendo una implementación en XML –, el protocolo BEEP (*Blocks Extensible Exchange Protocol*) aplicado a la detección de intrusos, y finalmente el protocolo IDXP (*Intrusion Detection eXchange Protocol*) para intercambio de información entre entidades de un sistema distribuido de detección de intrusos. Sin duda el primero y el último son especialmente importantes, ya que constituyen la base del sistema de intercambio de datos entre los diferentes elementos del entorno: marcan el 'lenguaje' que antes decíamos que tenían que saber hablar todas y cada una de las entidades del sistema distribuido.

Desde `http://www.ietf.org/html.charters/idwg-charter.html` pueden consultarse los trabajos del IDWG; se trata de un grupo activo, que realiza contínuamente revisiones y mejoras de sus borradores para tratar de convertirlos en un estándar real. Si algún día esto es así, habremos dado un paso muy importante en el diseño, implantación y gestión de sistemas distribuidos de detección de intrusos; lamentablemente, muchos de los productos actuales no parecen tener mucho interés en acercarse al estándar (quizás por miedo a perder cota de mercado). La integración de algunos sistemas (como SNORT) es bastante inmediata, pero en cambio la de otros – especialmente productos comerciales, altamente cerrados – no lo es tanto; mientras esto no cambie, es difícil que se consiga implantar el estándar, así que ójala estos fabricantes se den cuenta de que su adaptación a los borradores del IDWG produce sin duda más beneficios que daños.

Referencias

[And80] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Abril 1980.

[Axe98] Stefan Axelsson. Research in intrusion-detection systems: A survey. Technical Report 98–17, Chalmers University of Technology, Diciembre 1998.

[BB99] Roland Büschkes and Mark Borning. Transaction–based Anomaly Detection. In *Proceedings of Workshop on Intrusion Detection and Network Monitoring*. The USENIX Association, Abril 1999.

[CKL97] M. Ruschitzka C. Ko and K. Levitt. Execution monitoring of security–critical programs in distributed systems: A specification–based approach. In *Proceedings of the 1997* IEEE *Symposium on Security and Privacy*, pages 175–187. IEEE Computer Society, Mayo 1997.

[Com95] Douglas E. Comer. *Internetworking with* TCP/IP. *Volume 1: Principles, Protocols & Architecture*. Prentice Hall, 3rd edition, 1995.

[Con99] Intrusion Detection System Consortium. Intrusion Detection Systems buyer´s guide. Technical report, ICSA.NET, 1999.

[Esc98] Terry Escamilla. *Intrusion Detection: Network Security beyond the Firewall*. John Wiley and Sons, 1998.

[Fyo98] Fyodor. Remote OS detection via TCP/IP Stack Fingerprinting, Octubre 1998. http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[GL91] T.D. Garvey and Teresa F. Lunt. Model–based Intrusion Detection. In *Proceedings of the 14th National Computer Security Conference*, pages 372–385, Octubre 1991.

[Gra00] Robert David Graham. Network Intrusion Detection Systems FAQ v. 0.8.3, Marzo 2000. http://www.robertgraham.com/pubs/network-intrusion-detection.html.

[HLMS90] Richard Heady, George Luger, Arthur Maccabe, and Mark Servilla. The architecture of a Network Level Intrusion Detection System. Technical Report CS90–20, University of New Mexico, Agosto 1990.

[Ilg92] Koral Ilgun. USTAT: A real–time intrusion detection system for unix. In *Proceedings of the 1993 Symposium on Security and Privacy*, pages 16–28. IEEE Computer Society, Mayo 1992.

[ISV95] David Icove, Karl Seger, and William VonStorch. *Computer Crime. A Crimefighter's handbook*. O'Reilly & Associates, 1995.

[JV93] Harold S. Javitz and Alfonso Valdes. The NIDES Statistical Component: Description and Justification. Technical report, SRI International, Marzo 1993.

[Kem98] Richard A. Kemmerer. NSTAT: A Model–Based Real–Time Network Intrusion Detection System. Technical Report TRCS97-18, University of California, Junio 1998.

[Ko96] Calvin Cheuk Wang Ko. *Execution Monitoring of Security–Critical Programs in a Distributed System: A Specification–Based Approach*. PhD thesis, University of California at Davis, 1996.

[KS94] Sandeep Kumar and Eugene Spafford. An Application of Pattern Matching in Intrusion Detection. Technical Report CSD-TR-94-013, Purdue University, Marzo 1994.

[Kum95] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, Agosto 1995.

[L+92] Teresa F. Lunt et al. A real–time intrusion detection expert system (IDES). final technical report. Technical report, SRI International, Febrero 1992.

[Lis95] Justin Jay Lister. *Intrusion Detection Systems: an Introduction to the detection and prevention of computer abuse*. PhD thesis, University of Wollongong, 1995.

[Lun90] Teresa F. Lunt. Detecting Intruders in Computer Systems. In *Proceedings of the Sixth Annual Symposium and Technical Displays on Physical and Electronic Security*, 1990.

[Nor99] Stephen Northcutt. *Network Intrusion Detection: An Analyst´s Handbook*. New Riders, 1999.

[PK92] P.A. Porras and R.A. Kemmerer. Penetration state transition analysis: a rule–based intrusion detection approach. In *Proceedings of the 8th Computer Security Application Conference*, pages 220–229, Noviembre 1992.

[Por92] Phillip A. Porras. STAT*: A State Transition Analysis Tool for Intrusion Detection*. PhD thesis, University of California, Junio 1992.

[Ran98] Marcus J. Ranum. Intrusion Detection: Challenges and Myths. Technical report, Network Flight Recorder, Inc., 1998.

[Roe99] Martin Roesch. Snort – Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Systems Administration Conference – LISA´99*. The USENIX Association, Noviembre 1999.

[SG91] Shiuhpyng Winston Shieh and Virgil D. Gligor. A pattern–oriented intrusion model and its applications. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 327–342. IEEE Computer Society, Mayo 1991.

[Spi01] Lance Spitzner. Know your enemy: Honeynets. http://project.honeynet.org/papers/honeynet/, 2001.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated Volume I: The Protocols*. Addison Wesley, 1994.

[Sun96] Aurobindo Sundaram. An introduction to Intrusion Detection. *Crossroads: The ACM Student Magazine*, 2(4), Abril 1996.

[Tan96] Andrew Tanenbaum. *Computer Networks*. Prentice Hall, 1996.

[Thu00] Thuull. Anomaly Detection Systems. *2600: The Hacker Quartely*, 17(3), Primavera 2000.