Una propuesta para reducir las falsas alarmas en sistemas de detección de intrusiones basados en red para protocolo HTTP

Rodolfo Baader[1] Tomás Heredia[2]
Director: Dr. Patricia Borensztejn[3]

Julio de 2004

[1]rbaader@dc.uba.ar
[2]theredia@dc.uba.ar
[3]patricia@dc.uba .ar

Resumen

Para detectar intentos de ataque por Internet, y poder tomar las medidas necesarias para contrarrestar estos intentos o remediar sus efectos, se utilizan *Intrusion Detection Systems (IDS)*. Cada día los ataques representan un porcentaje mayor del tráfico en Internet. Según RipTech[1] y Symantec[2] la tasa de ataques del segundo semestre de 2002 creció un 20 % con respecto al mismo período del año anterior, y el crecimiento llega a un 102 % si se considera también la actividad de gusanos. Sumado a este problema, los IDSs generan falsas alarmas que engrosan los reportes de incidentes en un alto número, tornando en tedioso, costoso y poco útil el trabajo de analizarlos.

Se propone un método para mejorar el comportamiento de los IDS basados en reglas como el Snort[3], aplicado al protocolo HTTP. El mismo consiste en interpretar la respuesta del servidor ante las distintas peticiones. Se clasifican luego los intentos de ataques según el éxito del mismo, pudiendo así reducir la tasa de falsas alarmas.

Abstract

To detect attack attempts in Internet, and be able to take measures to stop them or correct their effects, *Intrusion Detection Systems (IDS)* are used. Every day, attacks represent a greater percentage of Internet traffic. According to Riptech[1] and Symantec[2] the attack rate in the second half of 2002, increased a 20% compared to the same period in 2001, and the grow scales to 102% if Internet worms are considered. Also false alarms are generated by IDSs that make reports grow in an important number, making the reviewing of Intrusion Detection alerts tedious, expensive and almost useless.

A method is proposed to improve behavior in rule based IDSs like Snort[3], applied to HTTP protocol. It consists of analyzing server response to the different requests. Attacks are then classified in wether they were successful or not, so the false alarm rate can be reduced.

Agradecimientos

Índice general

Capítulo 1

Introducción

Se denomina Detección de Intrusiones al proceso de monitorear los eventos que ocurren en un sistema o red de computadoras, analizándolos en busca de señales que indiquen incidentes de seguridad. Se puede pensar en sistemas análogos en otras áreas, como por ejemplo las alarmas anti-robo, o sistemas de vigilancia con videocámaras .

Los sistemas de detección de intrusiones (Intrusion Detection Systems o IDS) se pueden clasificar por diferentes características[4]. Según el ámbito donde se ejecutan, se pueden separar en "IDS de red" (NIDS), cuando inspeccionan el tráfico de red, e "IDS de host" (HIDS) cuando analizan cambios en los archivos, memoria o comportamiento de un equipo individual. Según la estrategia utilizada para detectar los ataques, se pueden distinguir los siguientes métodos: comparación con ataques conocidos, y detección de anomalías en el comportamiento. Cada una de estas estrategias tiene sus ventajas y desventajas.

Entre los defectos que presentan los sistemas existentes, podemos destacar:

- Imposibilidad de detectar si el ataque fue exitoso, generando un importante número de falsas alarmas.
- Una alta tasa de falsos positivos[5] (eventos normales que son considerados como ataques por el IDS)
- La existencia de herramientas para generar falsos positivos, con el objetivo de confundir a los IDS[6, 7] y, en consecuencia, dificultar la tarea del operador del sistema.
- Dificultad para detectar ataques no conocidos.

En el trabajo diario de un administrador de sistemas, el alto porcentaje de falsas alarmas generadas por los IDS dificulta la tarea de analizar los resultados brindados por estas herramientas. La existencia de grandes volúmenes de información irrelevante provoca que

los encargados de administrar estos sistemas tiendan a desperdiciar su potencial. Este fue el problema que motivó el desarrollo de este trabajo.

Acotando el espectro de la detección de intrusiones a un protocolo específico, se puede aprovechar la definición del mismo para distinguir las diferentes partes del mismo, su función, y su comportamiento normal. Se propone entonces un método para mejorar el comportamiento de los IDSs basados en reglas, aplicado al protocolo HTTP[1]. Al poseer información acerca del comportamiento normal de un servidor ante un intento de ataque dirigido a una vulnerabilidad específica, se puede interpretar la reacción del mismo. Con este método es posible clasificar los intentos de ataques según su éxito, y definir la atención que el mismo requiere. Al utilizar esta técnica se pueden disminuir significativamente las falsas alarmas.

El protocolo HTTP fue elegido para desarrollar este método por dos motivos:

- Según análisis de tráfico realizados por CAIDA[8], HTTP es uno de los protocolos más utilizados en Internet.
- Es un protocolo simple por diseño. A su vez, la complejidad de los servicios montados sobre el mismo (SOAP, WebDav, etc) fomentan la aparición de fallas que pueden provocar problemas de seguridad.

La elección del entorno sobre el que se desarrolla este trabajo también se vio influenciada por una incidencia del protocolo HTTP de más del 20 % sobre el total de los ataques sobre internet[9].

El IDS de red basado en reglas "Snort" fue elegido para comparar el método propuesto. Por ser un software con licencia GPL (libre disponibilidad) , goza de una gran popularidad. Por ejemplo, en una encuesta realizada por el sitio `http://www.secadministrator.com/` en octubre de 2002, el 91 % de los participantes respondieron que utilizaban Snort como herramienta de detección de intrusiones.

El trabajo se organiza de la siguiente manera: En el Capítulo 2 se introducen los conceptos necesarios sobre IDS y el protocolo HTTP. En el capítulo 3, se describe el estado del arte y la tecnología sobre el tema tratado. En el capítulo 4 se presenta la problemática tratada y se propone un modelo de solución, para luego validar el mismo en el capítulo 5, realizando pruebas con tráfico de red real y analizando los resultados. El capitulo 6 presenta las conclusiones y posibles trabajos futuros que podrían extender las ideas expresadas en este trabajo. Posteriormente, se incluyen apéndices conteniendo los ejemplos utilizados en laboratorio, la extensión propuesta de un lenguaje existente y el detalle del contenido del CD-ROM incluido con este trabajo.

[1] Extensiones al trabajo desarrollado en esta tesis incluyen la aplicación a otros protocolos

Capítulo 2

Conceptos

2.1. Sistema de Detección de Intrusiones

Los IDS recolectan información de diversos puntos de un sistema y la analizan para poder detectar comportamientos que puedan indicar ataques a la seguridad del mismo. La detección de intrusiones es tema de investigación desde la década del 80, y en 1988 Lunt[10] presentó el primer resumen sobre el tema. Si bien se cuenta con más de 20 años de desarrollo y con variadas implementaciones comerciales, aún quedan muchos problemas por resolver en este campo. Entre otros, podemos nombrar las falsas alarmas, problemas de escalabilidad y dificultad para detectar ataques no conocidos.

El uso de IDSs puede ofrecer muchos beneficios. Su principal función consiste en detectar intentos de intrusiones, y luego generar alertas que pueden utilizarse para mejorar la seguridad de un sistema informático. Estas mejoras pueden consistir en acciones, tanto manuales como automáticas, que bloqueen a los atacantes detectados o impidan que el ataque afecte la funcionalidad del sistema.

Es muy importante recordar que un IDS por si mismo no puede garantizar la seguridad de un sistema, y que debe formar parte de una estrategia de defensa completa que incluya otras tecnologías de seguridad informática.

2.1.1. Elementos y definiciones de un IDS

En la bibliografía[11, 12] se definen varios términos referentes a la detección de intrusiones y respuesta a incidentes. A continuación se presentan algunas definiciones útiles para este trabajo:

Ataque: Un ataque es un asalto a la seguridad de un sistema derivado de un acto delibe-

rado.

Evento: Un evento es una acción dirigida a un objetivo cuya intención es provocar un cambio de estado en el mismo.

Incidente: Un incidente es un evento que involucra una violación a la seguridad de un sistema.

Impacto: El impacto describe el resultado de un incidente expresado en términos de la comunidad. Por ejemplo según el costo financiero de la interrupción del servicio.

Objetivo: Un objetivo es una computadora o una entidad física o lógica relativa a un sistema de computación.

Víctima: La víctima es un individuo u organización que sufrió un incidente.

Vulnerabilidad: Una vulnerabilidad es una debilidad en un sistema que podría permitir el uso de acciones no autorizadas. Existen distintos tipos de vulnerabilidades y distintas formas de clasificarlas. Por ejemplo, teniendo en cuenta el recurso afectado, la falla existente o el impacto, según se describe en la sección 3.5.

2.1.2. Clasificación de IDS

Los Sistemas de Detección de Intrusiones pueden clasificarse según diferentes características, de las cuales se destacan dos: según su ámbito de ejecución y según el método de detección.

Según el ámbito de ejecución

Basados en host (HIDS): Son sistemas que se ejecutan en cada equipo, analizando los registros de auditoría (logs) generados por el sistema operativo y las aplicaciones. Realizando este análisis se pueden detectar accesos de usuarios locales desde lugares remotos, repetidos intentos fallidos de acceso al sistema, errores fatales de las aplicaciones, etc. Otra función que se encuentra en los HIDS consiste en verificar la integridad de archivos importantes del sistema para detectar cambios no autorizados. Para realizar esta tarea, preferentemente apenas instalado el equipo, se construye una base de datos con información sobre los archivos a proteger. En esta se almacenan datos como el tamaño, los permisos de acceso y la fecha de última modificación. Además, se genera un digesto digital seguro, pequeño identificador de un archivo o documento usualmente generado con algoritmos como MD5 o SHA-1 también conocido como hash, que tiene la propiedad de ser computacionalmente imposible encontrar dos documentos que originen un mismo digesto digital seguro. Este mecanismo se utiliza para detectar cambios en los archivos: si calculamos el hash de un archivo y lo

comparamos con el valor almacenado en la base de datos, se podrá determinar, sin posibilidad de error, si hubo modificaciones en dicho archivo. Una vez generada la base de datos, una aplicación compara periódicamente las características actuales de los archivos con la información almacenada en la base de datos, e informa sobre cualquier cambio en los atributos.

Basados en red (NIDS): Son equipos que se conectan a diversos segmentos de red y capturan todo el tráfico que pasa por los mismos, para poder analizarlo. Habitualmente, una placa de red (NIC) de una computadora conectada a una red Ethernet (IEEE 802.3), opera en modo no promiscuo. En este modo de operación, sólo las tramas dirigidas a la dirección de control de acceso al medio (MAC) específica de esa placa, más los paquetes *broadcast*, que están dirigidos a todos los equipos, son reenviados al sistema operativo para su procesamiento. El NIDS debe operar en modo promiscuo para poder monitorear todo el tráfico que pasa por el segmento de red al que se encuentra conectado, y no sólo el tráfico destinado a su propia dirección MAC. Generalmente, estos sistemas poseen dos placas de red. Una para *escuchar* el tráfico de red y otra para control y reporte de los ataques detectados. Al ser cada vez mas común el uso de redes switcheadas (que no permiten que la placa de red en modo promiscuo escuche todo el tráfico), los fabricantes de switches de red han comenzado a incluir la funcionalidad de replicar puertos para copiar todo el tráfico que se recibe en los puertos del dispositivo al puerto donde se conecta el NIDS.

Según el método de detección

Basados en Patrones conocidos: Son sistemas de detección de intrusiones que disponen de reglas que describen patrones de tráfico de red correspondientes a ataques conocidos, y utilizan estas reglas como base de comparación. Suelen ser eficientes para detectar ataques conocidos, pero no pueden detectar ataques que no presenten tráfico que se ajuste a pautas conocidas. Además, muchas veces se utilizan reglas demasiado estrictas que impiden detectar variaciones de un ataque conocido.

Basados en Heurísticas y/o probabilidades: Son sistemas de detección de intrusiones que intentan determinar, siguiendo heurísticas o métodos estadísticos, si hay acciones tendientes a atacar la seguridad de un sistema. Para lograr este objetivo, se intenta determinar inicialmente como es el tráfico normal. Luego, se intenta detectar desviaciones que podrían indicar un ataque. Tienen como principal ventaja el poder detectar ataques desconocidos. Sin embargo, es extremadamente dificultoso determinar qué tráfico de red debe considerarse “normal”.

Híbridos

Existen también sistemas híbridos, que utilizan distintos ámbitos de ejecución y/o distintos métodos de detección. Por ejemplo, existen sistemas como el Prelude (`http://www.prelude-ids.org`) que tienen agentes que funcionan como HIDS y y otros que funcionan como NIDS, y un módulo encargado de administrar las alertas generadas por los distintos agentes. Este "administrador" de alertas incluso puede en algunas implementaciones correlacionar eventos para poder tener un mayor grado de exactitud a la hora de determinar la existencia de un ataque. Otro ejemplo que combina diferentes técnicas es el Snort[3]. Éste es un IDS de red que detecta ataques en base a patrones, pero provee un preprocesador opcional llamado SPADE (Statistical Packet Anomaly Detection Engine) que permite detectar ciertas anormalidades en paquetes IP utilizando métodos estadísticos.

Honeypots

Los honeypots[13] (panales) son otro método utilizado para la detección de intrusiones. Estos sistemas funcionan emulando el funcionamiento de Sistemas Operativos y aparentan correr servicios vulnerables. Luego capturan información útil de los intentos de ataques que reciben. Todo tráfico dirigido a un honeypot es sospechoso, ya que dichos equipos no tienen servicios en producción y, por ende, no deberían ser utilizados por usuarios válidos.

Los honeypots son extremadamente útiles para analizar las conductas de los atacantes, entender y analizar los ataques perpetrados contra el sistema, y detectar ataques que eran previamente desconocidos.

2.2. Protocolo HTTP

El HyperText Transfer Protocol es un protocolo correspondiente al nivel de aplicación del modelo TCP/IP, utilizado para la navegación en forma distribuida de información hipermedial. Es el lenguaje que utilizan los clientes y servidores web para comunicarse entre sí. Fue definido inicialmente en 1990 por Tim Berners Lee, del laboratorio CERN. Es un protocolo simple, basado en texto, que no maneja estados. Esto significa que cada solicitud que el cliente envía al servidor es independiente de cualquier otra solicitud anterior. La naturaleza sin estados del protocolo permite a los usuarios navegar por la web siguiendo hipervínculos y visitando documentos (habitualmente llamados páginas cuando se utiliza el lenguaje HTML) en cualquier orden. Esto permite también distribuir las aplicaciones, ya que no se necesita correlacionar los pedidos, o inclusive replicarlas en múltiples servidores para balancear la carga generada por un número alto de solicitudes. La última versión, HTTP 1.1, se encuentra definida en el RFC 2616[14].

Para identificar recursos en los servidores web, se utilizan las URI[1][15]. Las mismas tienen la forma:

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

donde:

scheme: Es el protocolo que se usa. En caso de web puede ser http o https.

user: Usuario que puede ser requerido por el servidor para permitir el acceso al recurso.

password: Clave que puede ser requerida por el servidor para permitir el acceso al recurso.

host: Dirección IP o nombre completo (FQDN[2]) del servidor.

port: Número de puerto al cual conectarse (por defecto se utiliza el puerto correspondiente al protocolo designado en *scheme*, que en el caso de HTTP corresponde al puerto 80).

path: Sección que identifica el recurso dentro del contexto del servidor.

params: Usado en algunos casos para pasar parámetros del usuario.

query: Usado en algunos casos para pasar parámetros del usuario.

frag: Usado para indicar una parte del documento. Puede ser utilizado por el cliente para mostrar un fragmento en particular del recurso.

Es importante destacar que existen ciertas restricciones para escribir las URIs. Debido a las restricciones impuestas por la sintaxis del protocolo HTTP y de la gramática que define las URI, existen algunos caracteres que no pueden ser utilizados. Un ejemplo muy utilizado es el caracter *espacio*. Para poder utilizarlo, se debe reemplazar por la cadena *%20*. El caracter % indica que los siguentes dos caracteres deben interpretarse como dígitos hexadecimales, que representan un caracter en algún conjunto de caracteres, usualmente el US-ASCII. En este caso, 20 en hexadecimal corresponde a la posición que ocupa el caracter *espacio* en la codificación ASCII. De la misma manera, este mecanismo puede ser utilizado para codificar cualquier caracter. Esto implica que un mismo URI se puede escribir de distintas maneras. Por ejemplo, las URIs que se presentan a continuación identifican al mismo recurso:

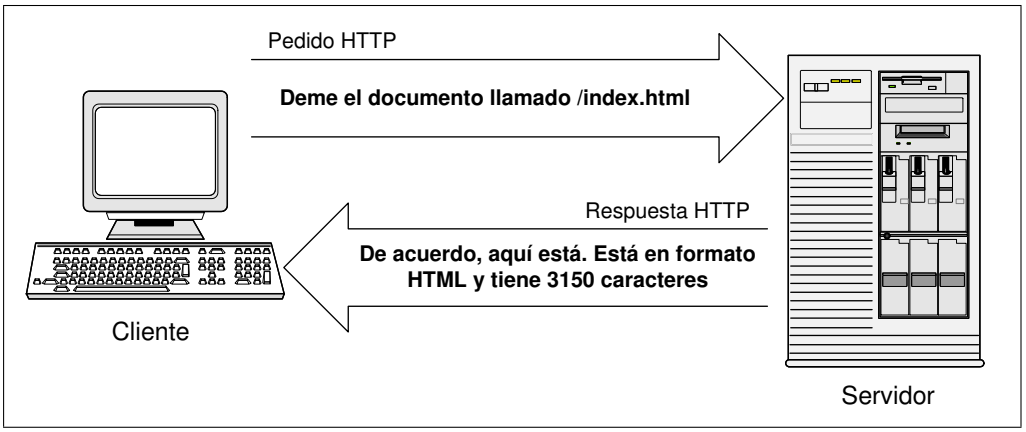
[1] Uniform Resource Identifier
[2] Full Qualified Domain Name

```
http://www.dc.uba.ar/people/misc/materias/
http://www.dc.uba.ar/fpeople/misc%2fmaterias/
http://www.dc.uba.ar/people/misc/m%61teri%61s/
```

El protocolo HTTP es un protocolo de pedido/respuesta (figura 2.1). Luego de establecer una conexión TCP, el cliente envía una solicitud al servidor. En dicha solicitud debe indicar el recurso al que quiere acceder , el método a través del cual lo accede y la versión de HTTP que va a utilizar. Luego, incluye una serie de encabezados que serán utilizados como modificadores. Opcionalmente, puede incluir un cuerpo adicional con contenidos para que sean procesados por el servidor.

En respuesta al pedido del cliente, el servidor responde con una línea de estado. En ésta se indica la versión del protocolo HTTP del mensaje, un código numérico de respuesta que indica éxito o fracaso, y un texto que describe dicho código para que el mismo pueda ser comprendido por el usuario. A continuación, el servidor envía una serie de encabezados en los que se incluye información del servidor e información de la entidad solicitada, posiblemente seguido por el contenido de la misma.

Figura 2.1: Ejemplo HTTP



A continuación se presenta un ejemplo de uso del protocolo HTTP. Una vez establecida la sesión tcp, el cliente envía un *Request HTTP* al servidor (figura 2.2).

Figura 2.2: Solicitud HTTP

```
GET index.html HTTP/1.0
```

Los métodos definidos en la versión 1.1 del protocolo HTTP son:

GET: Solicita la entidad identificada por el URI indicado en el pedido.

HEAD: De funcionalidad similar al GET, salvo que el servidor sólo debe devolver los encabezados, y no el contenido.

POST: Indica al servidor destino que reciba la entidad incluida en el request, para ser procesada según lo indicado en el URI del request.

PUT: Instruye al servidor para que almacene en el URI indicado, la entidad incluida en la solicitud.

OPTIONS: Solicita información acerca de los mecanismos de comunicación disponibles, métodos habilitados, etc.

DELETE: Solicita la eliminación del URI indicado en el servidor.

TRACE: Para que el servidor responda con una copia exacta de la solicitud que recibió. Se utiliza para rastreo de problemas.

CONNECT: Es utilizado por proxys para poder encapsular otros protocolos de comunicaciones en una conexión HTTP.

El método más comúnmente utilizado es el **GET**.

Siguiendo con el ejemplo de la figura 2.2, por la misma conexión tcp, el servidor enviará la respuesta (figura 2.3).

Figura 2.3: Respuesta HTTP

```
HTTP/1.1 200 OK
Date: Mon, 31 Mar 2003 13:00:34 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Content-Length: 1502
Connection: close
Content-Type: text/html; charset=ISO-8859-1

[DATA]
```

Los códigos de respuesta devueltos por el servidor en la línea de estado pueden agruparse en distintas categorías según su significado, como puede verse en la figura 2.4.

Los códigos que están en el rango 100-199 son utilizados para informar al cliente que puede continuar con la solicitud o para indicarle que debe cambiar a otro protocolo.

Figura 2.4: Códigos de respuesta HTTP

Code Range	**Response Meaning**
100-199	Informational
200-299	Client request successful
300-399	Client request redirected, further action necessary
400-499	Client request incomplete
500-599	Server errors

Los códigos que están en el rango 200-299 son los más comunes y son utilizados para indicar que la solicitud del cliente fue exitosa, y la respuesta del servidor contiene la información solicitada. El código más utilizado en este rango es el 200, que significa que la solicitud fue tratada exitosamente.

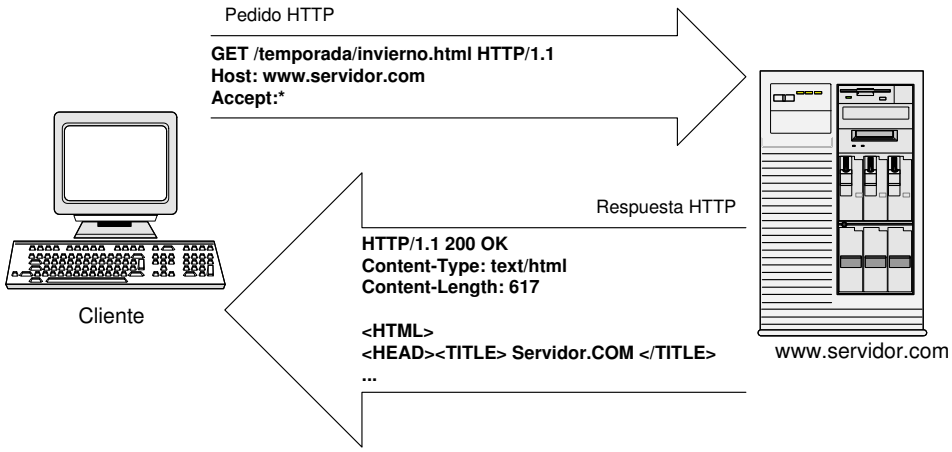
Los códigos que están en el rango 300-399 indican que se conoce al documento requerido, pero no se encuentra en el URI indicado. Permiten una redirección automática por parte del cliente indicándose la nueva ubicación. Además, pueden indicar si la redirección es temporal (código 307) o definitiva (código 301).

Los códigos que están en el rango 400-499 indican algún tipo de error en la solicitud del cliente. El más comúnmente encontrado es el 404, que indica que un recurso es inexistente. Este rango también puede indicar, entre otros estados, la falta de autorización (código 401) o prohibición de acceso (código 403).

Por último, los códigos que se encuentran en el rango 500-500 indican algún error en el servidor o en alguna aplicación invocada por el mismo.

En la figura 2.5 se ejemplifica un pedido completo HTTP versión 1.1 y su respuesta.

Figura 2.5: Ejemplo HTTP GET



Capítulo 3

Estado del Arte y Tecnología

La tecnología de los Sistemas de Detección de Intrusiones es inmadura y dinámica. Productos comerciales aparecen y desaparecen en forma frecuente, presentando mejoras más o menos importantes sobre sus predecesores. Los IDS han sido objeto de investigación por más de 20 años. A principios de los años 80, se planteaban las bases para crear un marco de trabajo para la detección de intrusiones[16]. Hoy, se siguen diversas líneas de investigación, principalmente orientadas a mejorar dos aspectos de la Detección de Intrusiones: la funcionalidad, mejorando la tasa de ataques detectados mediante la utilización de nuevos métodos y tecnologías, y la usabilidad, permitiendo que la gran cantidad de información generada pueda ser aprovechada.

En la sección 3.1 se resume el pasado y el presente en la investigación sobre IDS y falsas alarmas. A continuación, en la sección 3.2 se resume al análisis de protocolos como herramienta de detección de intrusiones, y en la sección 3.3 se muestran algunos Firewalls de aplicación para HTTP. En la sección 3.4 se describe el IDS de red Snort, que forma parte fundamental de este trabajo, como punto de partida para aplicar el método propuesto en el capítulo 4 y como base de comparación de resultados. En las secciones 3.5 y 3.6 se muestra una clasificación de eventos relacionados con ataques y algunos productos que incorporan este concepto. Por último, en la sección 3.7 se presentan diferentes lenguajes para la descripción de ataques, con énfasis en VulnXML, lenguaje a partir del cual se desarrolla este trabajo.

3.1. Trabajos relacionados

Hoy en día, gran parte de la investigación sobre IDS se centra en mejorar la tasa de falsas alarmas que presentan las implementaciones actuales, mejorando así la usabilidad de estos sistemas.

Algunos trabajos se orientan a la utilización de técnicas de detección de anomalías, generalmente basadas en estadísticas de tráfico. Usualmente se utilizan métodos de clasificación mediante técnicas de redes neuronales[17]. La detección de anomalías ha tenido éxito en la detección de ataques nuevos o desconocidos[18]. Sin embargo, debido a la dificultad en la definición o aprendizaje del tráfico *normal*, la tasa de falsas alarmas suele ser más alta que en otros métodos de detección de intrusiones.

Con un mejor comportamiento ante las falsas alarmas, se pueden citar a los sistemas basados en detección de uso erróneo (o no permitido). La técnica más utilizada consiste en la comparación del tráfico con patrones conocidos de uso malicioso. Kumar y Spafford[19] describen las características de los patrones utilizados para la detección de usos no permitidos.

Presentando un enfoque similar al de este trabajo, Kruegel[20] presenta un IDS utilizando diversas técnicas para detectar intrusiones en servidores y aplicaciones web. Posteriormente, Vigna et. al. [21] proponen un lenguaje llamado "*webstatl*", donde se analizan los requests del cliente y los logs de acceso al servidor.

Recientemente, y debido al fracaso en la disminución de la tasa de falsas alarmas, Martin Roesch, creador del Snort, anunció en una conferencia sobre seguridad informática del AusCERT[1] que las próximas versiones del Snort utilizarán un enfoque diferente para atacar este problema. Roesch indicó que se utilizarán técnicas de descubrimiento automático en forma pasiva de servicios existentes, tal como describe Grossman[22], para acotar el espectro de las comparaciones realizadas en el IDS.

También se ha trabajado en diversas clasificaciones y taxonomías referentes a la Detección de Intrusiones. Estas clasificaciones se refieren a diferentes aspectos de la tecnología, como ser terminología en el trabajo del "Incident Taxonomy and Description Working Group"[11], taxonomía de ataques en el trabajo publicado por Álvarez y Petrovic[23], o técnicas utilizadas por los IDS como describe Axelsson[4].

Otro enfoque al problema de las falsas alarmas es presentado por Cuppens[24]. Se indica que la creación de meta-alertas resumidas, mediante correlación de las alarmas originales, permite la consolidación de la información brindada por cada alerta disminuyendo el ruido provocado por una excesiva cantidad de las mismas.

3.2. Análisis de protocolos

El análisis de protocolos puede utilizarse para determinar si el uso de los protocolos es válido. Por ejemplo, para establecer una sesión TCP se inicia el TCP handshake cuando el cliente envía un paquete TCP con el flag SYN a un servidor. El servidor responde con un paquete con los flags SYN y ACK. A partir de este momento, todos los paquetes, tanto del

[1] `http://www.auscert.org.au/`

cliente como del servidor, tendrán el flag ACK encendido. Este proceso es conocido como *Three Way Handshake*.

Cualquier modificación al comportamiento "normal" de un protocolo es o bien un error, o un intento de "confundir" a algún sistema. Por este motivo el análisis de protocolos es utilizado para detectar ataques tanto conocidos como no conocidos[25].

Una variante del análisis de protocolos, es la *normalización*. Esta normalización puede realizarse en los protocolos del nivel de aplicación del modelo TCP/IP. Por ejemplo, el preprocesador *http_decode* de Snort puede normalizar diferentes codificaciones en los pedidos HTTP. Esta técnica también ha sido presentada para su utilización en los niveles de red y de transporte[26]

3.3. Firewalls de aplicación para HTTP

Este tipo de herramientas actúan en el nivel de Aplicación del modelo TCP/IP. Permiten aceptar o rechazar pedidos en base al contenido del tráfico HTTP. Por ejemplo, si reciben una solicitud mal formada o si el URL contiene una cierta cadena reconocida como un ataque, pueden impedir que la solicitud sea recibida por el servidor HTTP. El problema que presentan estas herramientas consiste en que que, en general, sólo chequean la información incluida en la solicitud del cliente, sin tener en cuenta la respuesta del servidor. Debido a esto, y según cómo se hayan escrito las reglas de filtrado, pueden llegar a impedir que llegue al servidor solicitudes válidas. Dentro de este grupo de herramientas, se pueden mencionar:

MS UrlScan: Herramienta desarrollada por la empresa Microsoft, se integra al servidor web MS Internet Information Server. Es un filtro ISAPI[2] que permite, mediante la configuración de un archivo de texto, definir métodos HTTP habilitados, extensiones de archivos prohibidas, cambio del Banner HTTP, entre otros.

Owasp CodeSeeker: Herramienta Open Source multiplataforma. Puede ser utilizada con una gran variedad de servidores web. Se instala en cada servidor, e intercepta el tráfico a nivel del Stack TCP/IP, luego del cifrado SSL y antes de ser recibida por la aplicación que brinda el servicio de web. Provee una consola de administración desarrollada sobre la plataforma Java, donde pueden configurarse las reglas de acción y visualizar reportes de las violaciones de seguridad, así como estadísticas varias. Esta consola permite administrar varios servidores en forma simultánea, y aplicar en todos una política uniforme.

ModSecurity: Herramienta open source que funciona como módulo de Apache. Permite

[2]Abreviación para **I**nternet **S**erver **API**, una API realizada por Microsoft para la programación de aplicaciones basadas en Web sobre el servidor MS IIS

filtrar solicitudes, y posee técnicas anti-evasión. Puede analizar el contenido transmitido utilizado el método POST de pedido del protocolo HTTP descripto en la sección 2.2, y puede loguear cada request en forma completa para su posterior análisis. Provee un script para convertir reglas de Snort al formato propio.

3.4. Snort

Snort es un IDS basado en patrones, y usa reglas para chequear paquetes que pasan por la red. Una regla es un conjunto de requerimientos que debe tener un evento (paquete, conjunto de paquetes reensamblados o flujo de datos), para disparar un alerta. Inicialmente fue desarrollado por Marty Roesch[3] únicamente como sniffer de paquetes para Linux, y su primer versión pública (0.96) data de diciembre de 1998. Sin embargo, fue evolucionando hasta convertirse en un IDS de red soportado en plataformas UNIX y Windows, con funciones equivalentes a las de los productos comerciales más maduros.

Actualmente, el Snort está formado por cuatro componentes básicos:

Sniffer: es el encargado de capturar todos los paquetes de la red. Utiliza la librería *libpcap* para realizar esta tarea.

Preprocesadores: permiten extender las funcionalidades preparando los datos para la detección. Se realizan tareas como la canonización de URLs, unión de porción de datos de los paquetes de una misma sesión o reensamblado de paquetes IP fragmentados. Existe un preprocesador muy útil relacionado con HTTP: el *http_decode*. Este preprocesador es utilizado para procesar las URIs contenidas en las solicitudes HTTP. Su principal función consiste en normalizar las URIs para que aparezcan en su forma más simple y unívoca. Como se describe en la sección 2.2, una misma URI puede escribirse de diversas maneras. Algunas técnicas de evasión de IDS se basan en escribir los URIs de un request HTTP de manera distinta para que el IDS no detecte el ataque, y en este hecho reside la gran utilidad del preprocesador *http_decode*.

Motor de detección: analiza los paquetes en base a las reglas definidas para detectar ataques.

Salida o Postprocesadores: permiten definir qué, cómo y dónde se guardan las alertas y los correspondientes paquetes de red que las generaron. Pueden utilizar archivos de texto, traps SNMP, bases de datos, syslog, etc.

[3]

Reglas de Snort

Las reglas de Snort están compuestas por dos secciones. El texto precedente al primer paréntesis, corresponde al encabezado. La sección entre paréntesis define las opciones. El encabezado de cada regla contiene:

- acción, normalmente *alert*
- protocolo
- direcciones IP origen y destino
- puertos de origen y destino

La sección de opciones contiene el mensaje que debe presentarse cuando se genere un alerta mediante dicha regla. Contiene además información específica de los patrones a buscar en los paquetes, junto con indicaciones de los lugares específicos dentro de un paquete donde deben encontrarse.

En la figura 3.1 se puede ver una regla simple de Snort. En la misma, se define que se debe generar un alerta si se encuentra un paquete con protocolo tcp, con origen en cualquier equipo y cualquier puerto, y destino en la red 192.168.1.0/24, puerto 111. Además el paquete debe contener la cadena hexadecimal "00 01 86 05". En caso de generarse el alerta, el mensaje enviado con el mismo será "mountd access".

Figura 3.1: Regla simple de Snort

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 05|"; msg:"mountd access";)
```

Las acciones disponibles para ser utilizadas en una regla de Snort son:

- alert - genera un alerta y guarda un registro del paquete.
- log - guarda un registro del paquete.
- pass - ignora el paquete.
- activate - genera un alerta y activa otra regla dinámica.
- dynamic - se mantiene sin uso hasta ser activada por otra regla con acción activate, luego funciona como una regla de log.

Figura 3.2: Regla de Snort

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (
 msg:"WEB-CGI HyperSeehsx.cgi directory traversal attempt"; uricontent:"/hsx.cgi";
content:"../../";content:"%00"; flow:to_server,established; reference:bugtraq,2314;
reference:cv,CAN-2001-0253; classtype:web-application-attack; sid:803; rev:6;)
```

En la figura 3.2 se presenta una regla de Snort más compleja, utilizada para detectar un ataque realizado utilizando el protocolo HTTP. En esta regla se indica que se debe generar un alerta cuando se encuentre un paquete perteneciente a una sesion TCP establecida. El paquete debe estar dirigido a un servidor web, y debe contener la cadena "/hsx.cgi" en la parte correspondiente al URI de la solicitud. Además, las cadenas "../../" y "%00" debe encontrarse en alguna otra parte de la solicitud HTTP. El mensaje a mostrar junto al alerta es: ""WEB-CGI HyperSeehsx.cgi directory traversal attempt". En la misma regla se incluyen: referencias a diferentes bases de datos donde se puede encontrar más información acerca de la vulnerabilidad, el tipo de ataque (en este caso, "web-application-attack"), el sid, o identificador único de reglas de Snort, y la versión de la regla, utilizada para diferenciar actualizaciones de la misma.

Para la clasificación de los ataques mencionada anteriormente, Snort define 34 clases de ataques (class-type). También permite definir diferentes prioridades según la clase de ataque a la que pertenece cada regla. Inclusive permite definir prioridades en forma individual para cada regla.

3.5. Taxonomía de ataques y Clasificación de eventos

El objetivo de una taxonomía es proveer una forma común de clasificar ataques. Actualmente, los ataques son descriptos de manera distinta por diferentes organizaciones. En su tesis de maestría, Daniel Lowry Lough [27] presenta un resumen de las propiedades que debe tener una buena taxonomía. A continuación se incluyen las más importantes:

Aceptada: La taxonomía debe ser estructurada para que pueda ser aprobada por la comunidad.

Comprensible: La taxonomía debe ser fácilmente comprensible para aquellos que trabajen en el campo de la seguridad informática y quienes tengan interés en ella.

Completa: Debe poder clasificar todos los tipos de ataques posibles.

Determinística: El procedimiento de clasificación debe estar claramente definido.

Mutuamente excluyente: Un ataque estará clasificado en una sola categoría.

No ambigua: Cada categoría de la taxonomía debe ser claramente definida para que no haya dudas acerca de en que categoría se clasifica un ataque.

Útil: Una taxonomía útil podrá ser utilizada en la industria de seguridad informática. Por ejemplo, la taxonomía debería poder ser utilizada por equipos de respuesta a incidentes.

A continuación se describe una taxonomía de ataques sobre protocolo HTTP desarrollada por Gonzalo Álvarez y Slobodan Petrovic[23].

Esta taxonomía se basa en utilizar el concepto de ciclo de vida de un ataque. Se utiliza la serie de pasos que realiza un atacante para llevar a cabo actividad maliciosa contra un servidor web. En cada etapa del ciclo de vida del ataque, se define el siguiente criterio de clasificación:

Punto de entrada: Distingue si el ataque se produce contra el servidor web o si se produce contra una aplicación que corre sobre dicho servidor.

Vulnerabilidad: Una vulnerabilidad es una debilidad en el sistema. Las mismas pueden ser clasificadas de la siguiente manera:

Inyección de código: Las vulnerabilidades de inyección de código permiten ejecutar en forma arbitraria código elegido por el atacante. Estas vulnerabilidades suelen existir debido a problemas de validación de datos de entrada en el código del servidor. En esta categoría se incluyen los ataques: Inyección de comandos de sistema, Inyección de comandos SQL y Cross Site Scripting, entre otros.

Manipulación de HTML: Este tipo de vulnerabilidad permite que un usuario malicioso modifique la información enviada entre el cliente (browser) y el servidor. Esto suele incluir la modificación de URLs, campos ocultos de formularios y Cookies.

Canonización: Estas vulnerabilidades se presentan cuando una aplicación toma decisiones de seguridad basándose en una cadena (nombre de archivo, directorio, URL) sin tener en cuenta que ese nombre puede ser expresado de distintas maneras. El ejemplo más común de estos ataques es el Directory Traversal Attack, que consiste en que el atacante acceda a datos que están fuera del directorio destino esperado.

Overflows: Un Buffer Overflow se produce cuando un programa escribe en posiciones de memoria que no estaban reservadas para variables. Generalmente sobreescribe código ejecutable, cambiando la secuencia lógica de ejecución del programa.

Problemas de configuración: Si la plataforma no está correctamente configurada, pueden aparecer varias vulnerabilidades. Por ejemplo, deben deshabilitarse las

cuentas de demostración y cambiar las claves por defecto, para evitar que un usuario no autorizado logre acceso al sistema utilizándolas.

Servicio bajo ataque: Existe un conjunto aceptado de servicios o requisitos que un sistema debe cumplir para ser considerado seguro. Esto incluye: Autenticación, confidencialidad, integridad, disponibilidad, control de acceso y auditoría.

Acción: Es la acción que realiza el ataque. Se distinguen tres objetivos distintos: Datos del servidor, autenticación de usuario y servidor web. Las acciones dirigidas contra los datos incluyen lectura, modificación, borrado e inserción. Las acciones dirigidas contra la autenticación incluyen la falsificación de identidad, el salteo de mecanismos de autenticación y la búsqueda de datos válidos de autenticación. Con respecto a las acciones dirigidas contra el servidor web, las mismas incluyen la interrupción (ej: Denegación de Servicio), pruebas para determinar las características del objetivo, u otras.

Longitud: La longitud de los datos de entrada puede ser útil para determinar si se intentó realizar un buffer overflow.

Método HTTP: Para que un ataque sea descripto usando esta taxonomía, debe utilizar el protocolo HTTP. Esto significa que debe utilizar un método HTTP como se describe en el capítulo 2.

Encabezado HTTP: Esta propiedad es opcional, y puede ser útil para identificar mejor ciertos ataques.

Objetivo: Algunos ataques pueden querer afectar exclusivamente una aplicación (afectar solo los datos y funcionalidad de la misma), mientras que otros pueden querer atacar a la plataforma entera, afectando información de otros servicios.

Alcance: Algunos ataques afectan a un usuario particular, mientras que otros tienen impacto en todos los usuarios del sistema.

Privilegios: Cuando un atacante compromete la seguridad de un servidor, puede obtener distintos niveles de acceso. Se puede distinguir entre usuarios no privilegiados y usuarios con permisos de administrador.

Con esta clasificación podemos determinar en forma precisa las propiedades de cada ataque, qué servicios afecta y cómo son afectados los mismos.

3.6. Productos relacionados

A continuación se presentan algunos productos que utilizan, en cierta medida, alguna clase de clasificación de ataques. El objetivo de dicha clasificación, tal como se anuncia

por las empresas fabricantes, consiste en focalizar la atención del operador en los eventos más importantes. En algunos casos, también se utiliza información referente al servidor o servicio atacado, siempre con el mismo objetivo: la disminución de las falsas alarmas.

- cisco Intrusion Protection
  `http://www.cisco.com/en/US/products/sw/secursw/ps2113/products_data_sheet09186a008017efb3.html.`
  Este producto anuncia la eliminación de falsas alarmas y escalamiento de ataques reales. Para alcanzar estos objetivos utiliza diversos métodos, incluyendo:
  - Investigación no intrusiva del objetivo para comparar sistema operativo y nivel de parches aplicados.
  - Análisis de logs y otros elementos en el equipo objetivo.
  - Recopilación de información forense: logs, logs de auditoría y pruebas de la intrusión.
- Mushroom Analyzer (MAN) (OpenSource) `http://man.sourceforge.net`.
  Se encuentra en etapa de planificación. Propone tres pasos simples:
  - Analizar el log de Snort y descartar las alertas que no sean interesantes en sí (Por ejemplo, un port scan).
  - Detectar si el objetivo es vulnerable mediante búsqueda de vulnerabilidades utilizando nessus.
  - Alertar al operador.
- eTrust Intrusion Detection
  `http://www.nwfusion.com/reviews/2001/1008bg.html`. Utiliza estadísticas para filtrar los falsos positivos.
- Snort clasifica las reglas con un valor numérico en el rango [1,10]. No está definido claramente el alcance de cada nivel, y la asignación del nivel a las diferentes reglas no sigue un patrón claro.

3.7. Lenguajes para descripción de ataques

Los lenguajes para descripción de ataques han recibido últimamente mucha atención por parte de la comunidad que investiga en temas sobre detección de intrusiones. Algunos ejemplos de lenguajes que permiten especificar vulnerabilidades son N-Code[28] , STATL[29] y VulnXML[30]. Si bien las reglas de Snort permiten identificar ataques sobre HTTP, se necesita para este trabajo un lenguaje más amplio, que permita especificar las vulnerabilidades, indicando a que plataformas afecta, y que reacción debe tener un servidor para ser considerado afectado por el ataque.

3.7.1. VulnXML

El lenguaje VulnXML es una especificación XML[31] que permite describir una vulnerabilidad de seguridad en un servidor o una aplicación web de manera no ambigua. Fue desarrollado por el Open Web Application Security Project (OWASP), proyecto creado en el año 2000 por un destacado grupo de profesionales en seguridad informática. La definición de tipos de datos (DTD) permite describir suficiente metadata sobre la vulnerabilidad para que un programa pueda, en forma automática, construir una o varias solicitudes HTTP. Estas solicitudes servirán para determinar si la vulnerabilidad existe en el equipo que está siendo examinado. El estándar es abierto, y se encuentran disponibles varios documentos XML de ejemplo que describen vulnerabilidades particulares.

Cada documento VulnXML describe una vulnerabilidad y la forma en la que una herramienta automatizada pueda, recorriendo todo un sitio, detectar las precondiciones necesarias para la ocurrencia de un ataque sobre la citada vulnerabilidad. En base a esas condiciones, armará un ataque para, en base a la reacción del servidor, discernir si la vulnerabilidad se encuentra presente en el equipo atacado. Cada documento está compuesto de tres partes y a continuación se describe cada una de ellas:

- Una descripción del test y la vulnerabilidad, en lenguaje coloquial.
- Declaración de variables.
- Descripción de los pasos necesarios para ejecutar el ataque.

La primer sección contiene información útil para que una persona comprenda de que se trata la vulnerabilidad, y algunos datos útiles para su ejecución. En la figura 3.3 puede verse un ejemplo de este elemento. Esta sección corresponde a un test que revisa la existencia de una vulnerabilidad presente en el servidor web Microsoft Internet Information Server, relacionada con un *desbordamiento de buffer* en el mecanismo de transferencia al utilizar *chunked encoding*.

Esta sección descriptiva, está compuesta compuesta de los siguientes atributos:

Name: Identificador único de la vulnerabilidad. Puede ser utilizado para referenciarla.

Version: Versión del documento. Debe ser actualizada cada vez que se publica un cambio en el documento.

Released: Fecha en el que se publicó por primera vez el el documento.

Updated: Fecha de última modificación. Las fechas deben estar en el formato definido por la RFC822.

Figura 3.3: VulnXML: elemento *TestDescription*

```
<TestDescription name="OWASP-00002" version="0.1" released="2002-04-10" updated="2002-04-30"
protocol="HTTP" affects="server" severity="high" alert="success"
type="Overflows" mayproxy="true">
  <Reference database="Bugtraq" URL="http://www.securityfocus.com/bid/4485"/>
  <Reference database="CVE" URL="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079"/>
  <Reference database="Microsoft" URL="http://www.microsoft.com/technet/security/bulletin/ms02-018.asp"/>
  <Reference database="Cert" URL="http://www.cert.org/advisories/CA-2002-09.html"/>
  <Copyright>Public Domain</Copyright>
  <Description>Buffer overflow in the chunked encoding transfer mechanism in Internet Information Server
       (IIS) 4.0 and 5.0 Active Server Pages allows attackers to cause a denial of service or execute
       arbitrary code.</Description>
  <ApplicableTo>
    <Platform>
      <OS>Windows</OS>
      <Arch>i386</Arch>
    </Platform>
    <WebServer>Microsoft-IIS</WebServer>
    <WebServer>Microsoft-IIS/[45].*</WebServer>
  </ApplicableTo>
  <TriggerOn event="file">
    <Match type="regex">.*asp</Match>
  </TriggerOn>
  <Impact>The attacker can cause the web server to crash and restart, and could potentially execute
       arbitrary code on the web server</Impact>
  <Recommendation>Delete sample ASP scripts to deter bulk scanners. Install the patch supplied by
       Microsoft as soon as it is available.</Recommendation>
</TestDescription>
```

Protocol: Protocolo utilizado para explotar la vulnerabilidad. Puede ser HTTP o alguno de los protocolos montados sobre él, como SOAP o WebDAV.

Mayproxy: Indica si el test puede ser realizado a través de un proxy, o sólo puede ser tratado mediante conexiones de red directas entre el cliente y el servidor.

Affects: Indica si el test afecta a todo el servidor, a un sitio virtual, a un directorio o a un archivo particular. Esta distinción permite indicar al encargado de ejecutar las pruebas si debe continuar intentando con la misma prueba sobre otros elementos del servidor, o si el resultado de una prueba es válido para todo el servidor, sitio virtual o directorio.

Severity: Gravedad de la vulnerabilidad. Puede ser baja (low), mediana (medium) o alta (high). Este atributo intenta ayudar al operador a evaluar la urgencia con la que debe tratar un alerta recibido.

Alert: Indica si debe generar un alerta cuando el test resultó positivo *(success)*, o negativo *(failure)*.

Type: Indica el tipo de ataque: Overflow, Validation, Configuration, u otros. Junto con el atributo severity, definen la taxonomía de ataques propuesta por VulnXML.

Reference: Referencias a fuentes de información externa sobre la vulnerabilidad. Por ejemplo, *Common Vulnerabilities and Exposures (CVE)*[4] y *Bugtraq ID (BID)*[5]. Las referencias incluyen URLs que apuntan a la información sobre la vulnerabilidad.

Copyright: Copyright y licencia de uso.

Description: Una descripción clara del ataque. Debería contener un resumen del mecanismo de ataque.

Applicable To: Servidor y Plataforma afectados. Puede ser útil al operador para simplificar la revisión de los resultados. Está compuesto por tres elementos:

Plataforma (Sistema operativo y arquitectura).

Producto utilizado como servidor web.

Producto utilizado como servidor de aplicaciones.

TriggerOn: Define condiciones bajo las que se dispara el ataque. Indica ante que elementos detectados en un servidor web, o una lista previa de URLs, se realiza un ataque con la definición dada por el documento. El atributo *event* define que parte del url es usada para disparar el ataque, y puede tomar los siguientes valores: authority, path, file o manual. El elemento *match* define una expresión regular que debe coincidir con el recurso existente para poder continuar con el análisis de esta vulnerabilidad. Este elemento es utilizado para la ejecución del test.

Impact: Descripción clara y resumida de los efectos que produce el ataque, en caso de ser efectivo.

Recommendation: Breve descripción de como hacer que un sistema o equipo no sea vulnerable a la falla descripta. Está dirigido al administrador del sistema o al desarrollador, dependiendo de la naturaleza de la vulnerabilidad.

Si se utilizara una herramienta automatizada de busqueda de vulnerabilidades con soporte para VulnXML y la misma tuviera la capacidad de recorrer todo un sitio web a través de los hipervínculos, el ejemplo de la figura 3.3 sería aplicado cuando la herramienta encontrase un archivo que contenga la cadena ".asp" en su nombre (elemento *TriggerOn*). Además, el chequeo sería realizado una única vez por servidor (atributo *affects*), y no por cada archivo que encuentre que concuerde con la cadena ".asp".

La segunda parte de un documento VulnXML corresponde a una primera definición de variables, utilizadas para modificar los tests a correr definidos en un <*Request*>, y no presenta particular interés para este trabajo.

[4] `http://cve.mitre.org/`

[5] `http://www.securityfocus.com/bid/`

Con respecto a la tercera parte, la misma corresponde a la descripción de las conexiones de red necesarias para probar la existencia de una vulnerabilidad en un servidor dado. Estas conexiones se encuentran descriptas por una serie de uno o más pasos.

Cada paso está compuesto por tres elementos:

- La solicitud del cliente (*Request*).
- La respuesta del servidor (*Response*).
- Un conjunto de uno o más criterios para determinar si el ataque fue exitoso en base a la solicitud y la respuesta recibida.

El elemento *Request* describe la solicitud enviada por el cliente al servidor web. Esto puede incluir encabezados, método HTTP, URIs y codificación utilizada, entre otros. En la figure 3.4 se presenta un ejemplo de una solicitud HTTP descripta en lenguaje VulnXML:

El elemento *Request* se encuentra divido en dos secciones:

- Encabezados, incluyendo la línea de pedido.
- Cuerpo de la solicitud. Según el método HTTP utilizado, éste será opcional.

A continuación, el elemento *Response* define un conjunto de variables en base al contenido de la respuesta a la solicitud descripta anteriormente. Las variables pueden tomar sus valores basados en la línea de status del servidor web, en los encabezados de la respuesta, o en base al contenido de la misma. El atributo *name* es el nombre de la variable, y debe seguir las convenciones de nombres de variables del lenguaje C. El atributo *type* define el tipo de datos, que en la versión actual sólo puede ser una cadena de caracteres. Cada variable puede tener una descripción que indica en forma coloquial el significado de la misma.

En la figura 3.5 se muestra un ejemplo de la definición de la variable *ResponseCode*. Aquí se le asigna como valor los tres dígitos que conforman el código HTTP devuelto por el servidor web en la línea de status de la respuesta que genera ante una solicitud.

En la figura 3.6 se presenta otro ejemplo de definición de una variable. En este caso se podría determinar, en base a la respuesta del servidor, si se aplicó la actualización que corrige la vulnerabilidad tratada.

Posteriormente a la definición de variables, mediante los elementos de *TestCriteria* se determina si el ataque fue exitoso o no. Para esto se realiza la comparación del contenido de las variables definidas en el elemento *Response* con los valores esperados según el equipo sea vulnerable o no.

En la figura 3.7 se presenta un ejemplo en el cual, en caso de ser ciertas las condiciones que se definen, se considera que la prueba falló.

Es decir, que el equipo en cuestión no es vulnerable. Debido a la forma de anidar los elementos *compare*, podemos distinguir entre condiciones lógicas *OR* y *AND*. En el ejemplo, el documento define que el resultado será *failure* si la variable *ResponseCode* es igual a 200 (código de status HTTP que indica que se devuelve un recurso en formal normal) Y la variable *body404*, que indica si se devuelve código 404 (file not found) en el contenido, no contiene dicho valor. También indica que la prueba no es exitosa si el *ResponseCode* es 302, 404, o 500.

Por último, en la figura 3.8 se muestra un ejemplo de un *TestCriteria* que indica que el ataque fue exitoso si la variable *unpatched* tiene un contenido distinto de vacío:

El DTD original de VulnXML, en su versión 1.4, puede ser consultado en el apéndice E. También se podrá encontrar en el anexo A el ejemplo completo de la vulnerabilidad de IIS que describimos parcialmente en este apartado.

3.7.2. Otros Lenguajes

El lenguaje N-Code es utilizado por el producto Network Flight Recorder[6] (NFR). Deriva de un lenguaje interpretado. Es un lenguaje de programación completo, que incluye instrucciones para control de flujo, procedimientos, variables con reglas de alcance, y tipos de datos complejos. Además posee tipos de datos adicionales a los de los lenguajes tradicionales, como el tipo de datos "dirección IP". El lenguaje puede ser utilizado para describir ataques producidos a través de una red.

STATL[29] es un lenguaje que permite realizar una descripción completa de un escenario de ataque. El ataque es modelado como una secuencia de pasos que llevan a un equipo de un estado inicial seguro a un estado final con la seguridad comprometida. El escenario debe tener por lo menos una transición y dos estados. El estado inicial no debe tener transiciones entrantes, y el estado final no debe tener transiciones salientes. Este lenguaje puede ser utilizado para describir ataques detectables por IDS basados en host y para IDS basados en red.

Se encuentran en desarrollo otros lenguajes para describir vulnerabilidades. Uno de ellos es el denominado AVDL[7] (Application Vulnerability Description Languaje), que busca crear un estándar interoperable para describir vulnerabilidades de seguridad en aplicaciones. Entre los trabajos actuales se destaca el WAS[8], desarrollado por el comité técnico en seguridad de aplicaciones web de OASIS[9]. El mismo se basa en VulnXML, proponien-

[6] `http://www.nfr.com/`
[7] `http://www.avdl.org`
[8] `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was`
[9] `http://www.oasis-open.org/`

do algunas mejoras. Representa la continuación del trabajo realizado sobre el lenguaje VulnXML, y se lo considera su sucesor.

Figura 3.4: VulnXML: *Request*

```
<Request>
   <MessageHeader>
     <Method encoding="text">POST</Method>
     <URI encoding="text">$scheme://$host:$port/$path/$file</URI>
     <Version encoding="text">HTTP/1.1</Version>
     <Header>
       <Name>Accept</Name>
       <Value>*/*</Value>
     </Header>
     <Header>
       <Name>Host</Name>
       <Value>$host</Value>
     </Header>
     <Header>
       <Name>Transfer-Encoding</Name>
       <Value>chunked</Value>
     </Header>
     <Header>
       <Name>Content-Type</Name>
       <Value>application/x-www-form-urlencoded</Value>
     </Header>
     <Header>
     <Header>
       <Name>Content-Length</Name>
       <Value>length=auto</Value>
     </Header>
   </MessageHeader>
   <MessageBody>
     <Separator encoding="text"/>
     <Item encoding="base64">MQpFCjAKCgoK</Item>
   </MessageBody>
</Request>
```

Figura 3.5: VulnXML: definición de variables en *Response*

```
<SetVariable name="ResponseCode" type="string">
    <Description>HTTP Response code</Description>
    <Source source="status">^.*\s(\d\d\d)\s</Source>
</SetVariable>
```

Figura 3.6: VulnXML: definición de variables en *Response*

```
<SetVariable name="patched" type="string">
    <Description>A patched server returns "(0x80004005)&lt;br&gt;Request</Description>
    <Source source="body">(\(0x80004005\)&lt;br&gt;Request)</Source>
</SetVariable>
```

Figura 3.7: VulnXML: *TestCriteria*

```
<TestCriteria type="failure">
    <ErrorMessage>The page was not found</ErrorMessage>
    <Compare variable="${ResponseCode}" test="eq">
      <Value>200</Value>
      <Compare variable="${body404}" test="neq">
        <Value>404</Value>
      </Compare>
    </Compare>
    <Compare variable="${ResponseCode}" test="eq">
      <Value>404</Value>
    </Compare>
    <Compare variable="${ResponseCode}" test="eq">
      <Value>302</Value>
    </Compare>
    <Compare variable="${ResponseCode}" test="eq">
      <Value>500</Value>
    </Compare>
</TestCriteria>
```

Figura 3.8: VulnXML: *TestCriteria*

```
<TestCriteria type="success">
    <Compare variable="${unpatched}" test="neq">
      <Value/>
    </Compare>
</TestCriteria>
```

Capítulo 4

Problemática y solución propuesta

4.1. Problemas en los IDS

Un problema importante que presentan los IDSs consiste en que generan falsas alarmas. En otras palabras, no pueden distinguir entre un ataque inofensivo de uno que afecte los sistemas monitoreados. Por esta razón, es necesario que un operador monitoree los alertas recibidos y se encargue de la decisión de accionar para contrarrestar o solucionar el ataque. Esta acción puede incluir el análisis de archivos de log, detección de cambios no autorizados en los servidores e identificación de paquetes dañinos, entre otros. Gracias a la proliferación actual de ataques de gusanos, o exploración de puertos abiertos, el problema se ve agravado por que la cantidad de falsos alertas generados puede ser muy alta, y, por ende, distraer al operador de los ataques reales. Es por eso que se plantea como objetivo buscar la manera de reducir el número de falsas alarmas. Debido a que se analizará la interacción entre los dos extremos de una comunicación, se consideró necesario basarse en un protocolo de aplicación específico, en lugar de utilizar herramientas genéricas que trabajen en la capa de red o de transporte del modelo ISO/OSI.

Para poder realizar un análisis adecuado, se debe distinguir entre falsas alarmas y falsos positivos.

Los falsos positivos son aquellos alertas generados por tráfico normal (que no es parte de un ataque). Muchas veces se pueden disminuir los falsos positivos haciendo más específicas las descripciones de los patrones de detección de una vulnerabilidad. Los falsos positivos son normalmente causados por una incorrecta definición de las reglas de detección del IDS.

Se consideran falsas alarmas a aquellas alertas generados en base a sesiones que forman parte de un ataque, pero donde el ataque no representa un peligro o violación a la seguridad. Este hecho puede deberse a varias razones, entre las que se destacan tres:

- El ataque afecta a una plataforma distinta a la atacada.

- La vulnerabilidad no existe en el objetivo atacado. Esto suele suceder cuando previamente se han tomado las medidas correctivas o paliativas necesarias, como por ejemplo la actualización del software vulnerable.
- El ataque no logró alcanzar el objetivo. Es común observar alertas generadas por IDSs que analizan el tráfico en un punto de la red que se encuentra en el camino (ruta) entre el atacante y el objetivo, pero este último se encuentra protegido por dispositivos u otras medidas de seguridad (por ejemplo un firewall).

En la bibliografía también se utiliza el término ruido para describir las falsas alarmas[32].

Por otra parte, se puede hablar de falsos negativos, cuando se producen ataques y los mismos no son detectados por el IDS. El problema es similar a lo que ocurre con los productos antivirus, que en general no pueden detectar un nuevo virus hasta que el fabricante no distribuye la actualización que contiene el patrón de reconocimiento de dicho virus. Para atenuar este problema, los IDS pueden utilizar técnicas estadísticas[33].

Como se planteó anteriormente, una de las grandes falencias de los IDSs es la generación de una gran cantidad de alertas, sin distinguir el impacto que producen los distintos ataques. Además de no permitir identificar si el ataque fue exitoso o no, tampoco indican la gravedad del ataque. Esto dificulta la tarea posterior de análisis por parte del operador del sistema. La utilización de una buena taxonomía de ataques puede colaborar con la solución a este problema. Esta permitiría identificar, en caso de que el ataque haya sido exitoso, si se realizaron cambios en los datos, si se ejecutó código arbitrario, a que tipo de permisos tuvo acceso el atacante, entre otros datos. De esta manera, el operador del sistema de detección de intrusiones podría recibir las alertas ordenadas según la prioridad dada por el impacto del ataque.

4.2. Propuesta

Se propone una técnica consistente en tener en cuenta la información completa de las sesiones HTTP, y no sólo las solicitudes generadas por los clientes. Se analizarán los detalles de la respuesta del servidor, tanto el código de error de la respuesta, como el contenido de la misma. Con estos datos se intentará determinar si el ataque fue exitoso o no. Esto permitirá lograr el objetivo principal de este trabajo: reducir considerablemente el número de falsas alarmas.

Se utilizó entonces un lenguaje para describir las vulnerabilidades conocidas sobre el protocolo HTTP, y se implementó un prototipo para demostrar la viabilidad del modelo propuesto, tal como se describe en la sección 4.4. Para describir las vulnerabilidades se utilizó una modificación del lenguaje VulnXML, que se presenta en la sección 4.3 En la sección 4.5 se muestran los resultados de la aplicación del prototipo en algunas pruebas de

laboratorio, y en el capítulo 5 se muestra la efectividad de la propuesta realizando pruebas que utilizan capturas de red de tráfico real obtenido externamente.

4.3. VulnIdsXML: una extensión al VulnXML

Para definir las reglas que detecten los ataques, se propone una extensión al lenguaje VulnXML, que denominamos VulnIdsXML. El lenguaje VulnXML fue diseñado para definir los datos necesarios para realizar un ataque con el objetivo de determinar la vulnerabilidad de un servidor, servicio o aplicación web a un problema conocido determinado. En este trabajo se extendió este lenguaje para detectar ataques mediante la inspección de tráfico.

Se modificó el DTD de la versión 1.4 de VulnXML (ver anexo E) para incluir un nuevo elemento similar al *Request* de una conexión, el cual se identificó como *IDSDetectRequest*. Este nuevo elemento utiliza una sintaxis similar a la de su antecesor, pero difiere totalmente en la semántica. En este elemento se definen propiedades más genéricas de la vulnerabilidad que en el anterior, para que se puedan detectar las distintas instancias posibles de un ataque. De esta manera, se incorpora el concepto de regla de detección correspondiente a un IDS basado en reglas. Continuando con el ejemplo presentado en la sección 3.7.1 donde se describe la sintaxis del lenguaje VulnXML, en la figura 4.1 se muestra el elemento *IDSDetectRequest* correspondiente a la extensión propuesta.

Figura 4.1: IDSDetectRequest

```
<IDSDetectRequest>
     <MessageHeader>
       <Method/>
       <URI encoding="text">.*\.asp</URI>
       <Version/>
       <Header>
         <Name>Transfer-Encoding</Name>
         <Value>chunked</Value>
       </Header>
     </MessageHeader>
</IDSDetectRequest>
```

Para concordar con la especificación de este documento XML, el URI de la solicitud HTTP examinada deberá concordar con la expresión regular ".*\.asp". Por omisión, también se indica que el método HTTP utilizado no es de importancia, y cualquiera será aceptado. Además, deberá existir en la solicitud HTTP el encabezado *Transfer-Encoding: chunked*. Si una solicitud examinada coincide con estas características, se habrá detectado un

intento de ataque sobre la vulnerabilidad descripta por el documento XML, y se deberán evaluar los criterios de prueba (*TestCriteria*) para determinar si el ataque fue exitoso.

En la figura 4.2 se puede ver el contenido de un pedido HTTP que coincide con el *IDSDetectRequest* especificado. El texto resaltado indica las secciones donde coincide con los patrones requeridos en el documento VulnIdsXML.

Figura 4.2: Ejemplo de Ataque

```
POST http://www.server.com/applicacion/appl.asp HTTP/1.1
Acept: */*
Host: www.example.com
Transfer-Encoding: chunked
Content-Type: application/x-www-form-urlencoded
Content-Length: length=auto

[DATA]
```

El fragmento incorporado al DTD de VulnXML correspondiente a la extensión propuesta se puede consultar en el anexo B.

4.4. Desarrollo del Prototipo

Una vez definido el lenguaje de descripción y detección de ataques, se desarrolló un prototipo para comparar los alertas obtenidos mediante el IDS Snort con los obtenidos por el modelo propuesto, y comprobar empíricamente su efectividad. El prototipo se desarrolló en lenguaje Perl[1], con el agregado de los módulos[2] necesarios para el manejo de paquetes TCP/IP, canonización de URIs y manejo de XML.

El programa recibe dos parámetros: el nombre de un archivo que contiene la captura del tráfico de red correspondiente a una sesión HTTP, y el nombre de un documento VulnIdsXML que describe una vulnerabilidad supuestamente atacada por el tráfico incluido dentro del archivo. El archivo de entrada debe seguir el formato PCAP[3], utilizado por Snort y la mayoría de los programas GNU que manejan capturas de paquetes de red. Este archivo debe contener únicamente los paquetes correspondientes a una sesión HTTP. La primer parte del prototipo está dedicada a extraer el stream HTTP de los paquetes contenidos en el archivo de captura definido como parámetro. Se extraen tanto la solicitud del cliente como

[1] *Practical Extraction and Reporting Language*, `http://www.perl.org`
[2] obtenidos en `http://cpan.org`
[3] `http://www.tcpdump.org`

la respuesta del servidor. Se analizan brevemente los flags de los paquetes TCP para poder detectar los distintos pasos del *three-way handshake*, mecanismo por el cual se establecen las conexiones TCP. Esto permite identificar cuál es la dirección IP correspondiente al cliente, y cuál es la que corresponde al servidor. Una vez determinados los IPs del cliente y servidor, se puede distinguir entre el tráfico proveniente de cada uno.

Debido a que el objetivo del desarrollo del prototipo es únicamente validar el modelo, se decidió simplificar la implementación suponiendo que el archivo de entrada contiene el conjunto de paquetes que componen únicamente la sesión TCP completa, sin duplicados, con los paquetes ordenados y en forma completa. En un producto más evolucionado, se deberían eliminar estos presupuestos y utilizar técnicas más adecuadas para detectar las sesiones. Estas técnicas son comúnmente utilizadas por una gran variedad de productos como Snort[34] y tcptrace[4].

Una vez identificadas la solicitud del cliente y la respuesta del servidor contenidas en la captura, se comienza a procesar el documento VulnIdsXML conteniendo la definición de la vulnerabilidad y su forma de detección. Se verifica a continuación que la solicitud del cliente se corresponda con el patrón definido en el elemento *IDSDetectRequest* y se le asignan valores a las variables definidas en el documento XML según los datos contenidos en la respuesta del servidor. Finalmente, se aplican los tests descriptos en el documento VulnIdsXML. Estos consisten en predicados de lógica de primer orden que evalúan y comparan el contenido de las variables con valores predefinidos. Esto permite determinar si el ataque presente en la sesión HTTP fue exitoso o no. Para evaluar este resultado se definió una función recursiva, ya que en la evaluación de la respuesta es posible encontrar predicados complejos que incluyen varios niveles de anidamiento de condiciones lógicas.

De la evaluación de los tests definidos en el documento VulnIdsXML puede llegarse a los siguientes resultados:

SUCCESS: alguno de los tests definió que el ataque fue exitoso.

FAILURE: alguno de los tests definió que el ataque NO fue exitoso.

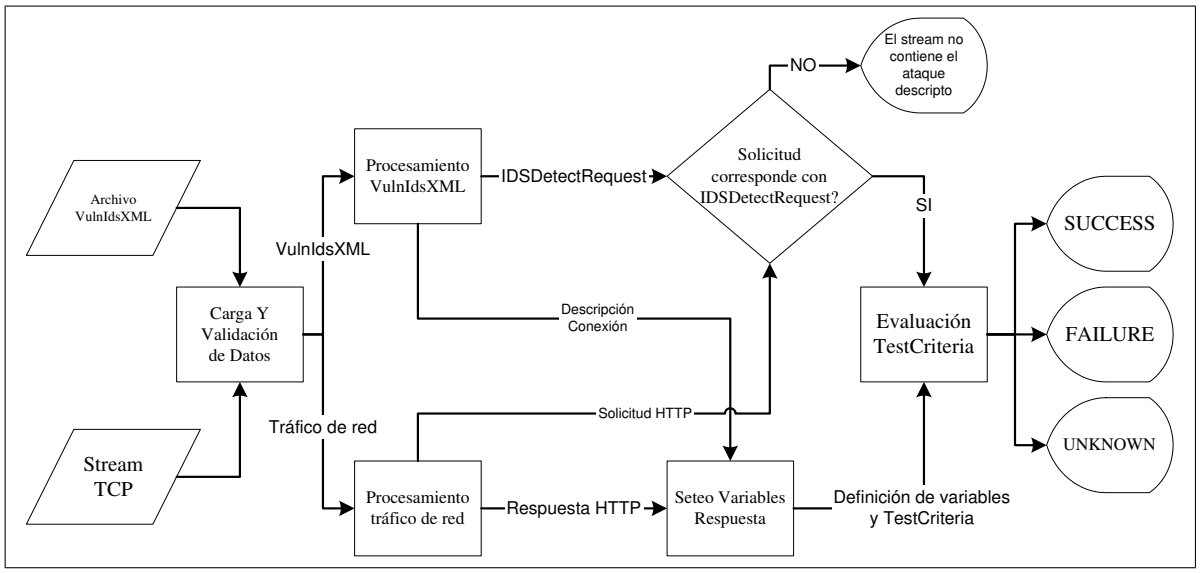
UNKNOWN: ninguno de los tests pudo definir con certeza si el ataque fue exitoso o no. El alerta no puede ser ponderado.

En la figura 4.3 se puede observar un esquema del funcionamiento del prototipo, mediante la utilización de un diagrama de flujo.

Adicionalmente al prototipo, se desarrolló una pequeña herramienta que procesa un documento en formato VulnIdsXML, y muestra por pantalla la información relacionada con la descripción de alto nivel de la vulnerabilidad, el tipo de ataque y la gravedad, impacto, recomendaciones y referencias a bases de datos de vulnerabilidades con más información,

[4]http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html

Figura 4.3: Diagrama de Flujo del Prototipo



en un formato fácilmente legible. En la figura 4.4 se muestra la salida que genera esta aplicación, correspondiente a la vulnerabilidad del ejemplo de la figura 4.1.

Figura 4.4: Descripción de una Vulnerabilidad

```
Vulnerability Description
-------------------------
OWASP-00002      type: Overflows         severity: high  Date: 2002-04-10

Buffer overflow in the chunked encoding transfer mechanism in Internet Informati
on Server (IIS) 4.0 and 5.0 Active Server Pages allows attackers to cause a deni
al of service or execute arbitrary code.

Impact: The attacker can cause the web server to crash and restart, and could po
tentially execute arbitrary code on the web server

Recommendation: Delete sample ASP scripts to deter bulk scanners. Install the pa
tch supplied by Microsoft as soon as it is available.

References:
-----------
Microsoft <http://www.microsoft.com/technet/security/bulletin/ms02-018.asp>
Bugtraq <http://www.securityfocus.com/bid/4485>
CVE <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079>
Cert <http://www.cert.org/advisories/CA-2002-09.html>
```

4.5. Prueba del prototipo

En esta sección se muestran algunas pruebas realizadas con cuatro vulnerabilidades de servidores y aplicaciones web. Se muestra el comportamiento de Snort y del prototipo que implementa el modelo presentado.

Se utilizaron cuatro vulnerabilidades para realizar las pruebas: tres que afectan diferentes bugs del servidor web *Microsoft Internet Information Server*, y una para un bug genérico de falta de validación de entradas y ejecución del comando *id* de Unix. Este comando muestra las propiedades (nombre, identificador, etc.) del usuario que ejecuta el proceso que lo invoca. En este caso, se trata del usuario que ejecuta el servidor web.

4.5.1. Pruebas

Para experimentar la reacción de los dos sistemas ante ciertos ataques, se armó un laboratorio donde se instalaron tres equipos con diferentes sistemas operativos, con diferentes niveles de actualización de los mismos. Un cuarto equipo realizó los ataques, y guardó las sesiones de red en archivos en formato PCAP, mediante la utilización de un *sniffer*. Esas capturas fueron luego utilizadas como datos de entrada para ejecutar el Snort y el prototipo.

Para cada uno de los ataques correspondientes a las tres vulnerabilidades que afectan plataformas Windows se realizaron pruebas contra tres equipos diferentes cuyas características principales son:

- Equipo 1: Windows 2000 Español + IIS5
- Equipo 2: Redhat 7.3 + apache 1.3.27
- Equipo 3: Windows 2000 Español + IIS5 con algunas actualizaciones de seguridad

Para el caso de la detección del intento de ejecución del comando *id*, se utilizó la aplicación *php-ping*. Esta aplicación, desarrollada en php, permite ejecutar el comando *ping* del sistema operativo. Como destino del *ping*, se utiliza una dirección provista en la sección de parámetros del URI con que se invoca a la aplicación. Como el programa *php-ping* no valida los datos de entrada que recibe, es posible la utilización de un IFS[5] para ejecutar comandos del sistema. Para este caso, se evaluaron tres escenarios en el equipo número 2:

- Servidor web Apache con el script vulnerable.
- Servidor web Apache con el script corregido.
- Servidor web Apache sin el script.

Descripción de los ataques

El primer caso de vulnerabilidad explotada corresponde a la vulnerabilidad referenciada por el boletín MS00-044 de Microsoft, CVE-2000-0630 de CVE y la regla 1725 de Snort. Este ataque consiste en adicionar la cadena "+.htr" a un URI válido. En un servidor vulnerable, esto puede permitir obtener fragmentos de código fuente normalmente no accesibles. En la figura 4.5 puede verse la regla correspondiente de Snort para detectar el ataque.

Figura 4.5: Reglas de snort: IIS +.htr code fragment

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS +.htr code fragment attempt";
flow:to_server,established; uricontent:"+.htr"; nocase; reference:cve,CVE-2000-0630;
classtype:web-application-attack; sid:1725;  rev:3;)
```

El segundo caso estudiado corresponde al boletín MS00-078, CVE-2000-0884 y a la regla 1945 de Snort. Consiste en utilizar una doble codificación del caracter /"para sortear las restricciones de acceso a directorios y listar o ejecutar archivos fuera del directorio raíz del servidor web. En la figura 4.6 puede verse la regla correspondiente de Snort para detectar el ataque.

[5]Input Field Separator

Figura 4.6: Reglas de snort: IIS unicode directory trasversal

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS unicode directory traversal attempt"; flow:to_server,established;
content:"/..%255c.."; nocase; classtype:web-application-attack; reference:cve,CVE-2000-0884;
sid:1945; rev:1;)
```

La última vulnerabilidad del IIS estudiada corresponde al boletín MS01-023, CVE-2001-0241 y a la regla de Snort 971. Consiste en aprovechar un desbordamiento de buffer en la extensión *Internet Printing ISAPI*. Esta extensión permite realizar impresiones a través de la web, y está incluida en la instalación por defecto del servidor Internet Infomation Server 5.0, que forma parte del sistema operativo Microsoft Windows 2000. En la figura 4.7 puede verse la regla correspondiente de Snort para detectar el ataque.

Figura 4.7: Reglas de snort: IIS ISAPI .printer

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS ISAPI .printer access";
uricontent:".printer"; nocase; flow:to_server,established; reference:cve,CAN-2001-0241;
reference:arachnids,533; classtype:web-application-activity; sid:971;  rev:3;)
```

La cuarta vulnerabilidad utilizada corresponde a un caso genérico de ejecución de código, en la que Snort detecta la presencia de la cadena ";id"en un URL. Esto suele indicar un intento de utilizar un IFS (;), y la ejecución del comando de Unix (*id*). Este comando es inofensivo, ya que no realiza ningún cambio en el estado del servidor, pero es útil para la recolección de información previa a la ejecución de un ataque. En la figura 4.8 pueden verse las reglas correspondiente de Snort para detectar el ataque y su ejecución exitosa.

Figura 4.8: Reglas de snort: comando *id*

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS id command attempt";
flow:to_server,established; content:"\;id";nocase; sid:1333; classtype:web-application-attack;
rev:4;)

alert tcp $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any
(msg:"ATTACK RESPONSES id check returned www"; flow:from_server,established; content:"uid=";
content:"(www)"; classtype:bad-unknown; sid:1882; rev:2;)
```

4.5.2. Generación de las reglas para el prototipo

Para poder procesar los alertas con el prototipo, y comparar su resultado con el comportamiento de Snort, se escribieron los documentos VulnIdsXML correspondientes a las

vulnerabilidades mencionadas. Para realizar esta tarea, se tomaron las reglas de Snort como punto de partida para definir los elementos *IDSDetectRequest*. Luego, se realizaron ataques a cada servidor, sabiendo a priori si eran vulnerables o no. Observando el comportamiento del servidor en cada caso, y junto a información detallada de los ataques obtenida en diversos sitios de seguridad accesibles por Internet, se definieron las variables relacionadas con el elemento *Response* y los *TestCriteria* adecuados para ponderar el alerta tratado.

Por ejemplo, en el caso de la vulnerabilidad descripta en el boletín MS00-044, se definió como variable el código de error HTTP devuelto por el servidor, y basta analizar si la misma lleva el valor *200* para saber si el ataque fue exitoso o no. En la figura 4.9 puede verse la definición de variables y los criterios de evaluación para esta vulnerabilidad.

Figura 4.9: Definición de variables y de Criterios de evaluación

```
<Response>
  <SetVariable name="ResponseCode" type="string">
    <Description>HTTP Response code</Description>
    <Source source="status">^.*\s(\d\d\d)\s</Source>
  </SetVariable>
</Response>
<TestCriteria type="success">
  <ErrorMessage>Request Successfull</ErrorMessage>
  <Compare variable="${ResponseCode}" test="eq">
    <Value>200</Value>
  </Compare>
</TestCriteria>
<TestCriteria type="failure">
  <ErrorMessage>request unsuccessful</ErrorMessage>
  <Compare variable="${ResponseCode}" test="neq">
    <Value>200</Value>
  </Compare>
</TestCriteria>
```

Los documentos VulnIdsXML generados se incluyen en el anexo C.

4.5.3. Comportamiento de Snort

Utilizando las capturas de tráfico de red descriptas en la sección 4.5.1, se ejecutó el programa Snort para observar sus resultados. En todos los casos planteados, tanto en los intentos de ataque a servidores vulnerables como en los no vulnerables, Snort generó el alerta correspondiente a la regla mencionada. En el caso de la ejecución del comando *id*, Snort

generó además un alerta adicional al reconocer la salida del comando *id* (`uid=48(apache)`) en la respuesta HTTP del equipo que contenía el script vulnerable.

En la figura 4.10 se presenta a modo de ejemplo el alerta generado por Snort ante la detección de un ataque correspondiente a la vulnerabilidad descripta en el boletín MS00-044 de la empresa Microsoft.

Figura 4.10: Alerta de Snort

```
[**] [1:1725:3] WEB-IIS +.htr code fragment attempt [**]
[Classification: Web Application Attack] [Priority: 1]
07/02-23:57:48.722174 xxx.xxx.xxx.xxx:1026 -> xxx.xxx.xxx.xxx:80
TCP TTL:64 TOS:0x0 ID:9359 IpLen:20 DgmLen:178 DF
***AP*** Seq: 0x705C9BCB  Ack: 0xDED6FD25  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 180042 0
[Xref => cve CVE-2000-0630]
```

En la primera línea de este registro de alerta se indican:

- Identificador de la regla de Snort (SID). En este caso, 1725.
- Versión de la regla (3).
- Descripción de la regla.

En la segunda línea se presentan:

- Clasificación del ataque (Web Application Attack)
- Prioridad de atención del evento.

En todos los casos, estos datos corresponden a la definición de la regla de Snort. En las siguientes 4 líneas se describen los datos del paquete IP que generó el alerta. Y por último, se presentan las referencias a bases de datos de vulnerabilidades, tal como se describe en la regla de Snort correspondiente.

En la figura 4.11 pueden verse los alertas generados ante las restantes vulnerabilidades.

Figura 4.11: Alertas de Snort

```
[**] [1:1945:1] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
07/02-23:54:02.843833 xxx.xxx.xxx.xxx:1025 -> xxx.xxx.xxx.xxx:80
TCP TTL:64 TOS:0x0 ID:32304 IpLen:20 DgmLen:252 DF
***AP*** Seq: 0x631A85AA  Ack: 0xDB5797D4  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 64393 0
[Xref => cve CVE-2000-0884]

[**] [1:971:3] WEB-IIS ISAPI .printer access [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
07/03-00:05:08.666511 xxx.xxx.xxx.xxx:1030 -> xxx.xxx.xxx.xxx:80
TCP TTL:64 TOS:0x0 ID:6677 IpLen:20 DgmLen:1234 DF
***AP*** Seq: 0x8D4A3540  Ack: 0xE594AA6E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 405294 0
[Xref => arachnids 533][Xref => cve CAN-2001-0241]

[**] [1:1333:4] WEB-ATTACKS id command attempt [**]
[Classification: Web Application Attack] [Priority: 1]
07/02-16:21:20.616549 xxx.xxx.xxx.xxx:2441 -> xxx.xxx.xxx.xxx:80
TCP TTL:128 TOS:0x0 ID:20985 IpLen:20 DgmLen:415 DF
***AP*** Seq: 0x2AE01EBC  Ack: 0xB2A8FB02  Win: 0xFAF0  TcpLen: 20

[**] [1:1882:9] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/02-16:21:20.646082 xxx.xxx.xxx.xxx:80 -> xxx.xxx.xxx.xxx:2441
TCP TTL:64 TOS:0x0 ID:57295 IpLen:20 DgmLen:513 DF
***AP*** Seq: 0xB2A8FB02  Ack: 0x2AE02033  Win: 0x1920  TcpLen: 20
```

En total, Snort generó 13 alertas.

4.5.4. Comportamiento del Prototipo

Una vez concluidas las ejecuciones de Snort, se utilizaron los documentos VulnIdsXML descriptos en la sección 4.5.2 para verificar el correcto funcionamiento del prototipo.

En los dos primeros ataques contra el servidor web Microsoft IIS y el ataque genérico aplicable a servidores Unix, el prototipo detectó correctamente los casos en que el servidor no era vulnerable. Para los casos en los que el ataque no fue exitoso, la respuesta del prototipo fue FAILURE, y para los casos en que el ataque funcionó, se anunció SUCCESS.

En el caso de la vulnerabilidad relacionada con el *Internet Printing ISAPI* del servidor web Microsoft IIS, el prototipo pudo determinar que el ataque no fue exitoso ante la respuesta del servidor Apache. En los otros dos casos, el prototipo no pudo determinar el

éxito o el fracaso de los ataques utilizando las verificaciones definidas en el correspondiente documento VulnIdsXML. Esto se debió a que en ambas instancias de IIS, el servidor cortó la conexión TCP sin devolver la correspondiente respuesta HTTP, siendo ésta la que contiene los datos que utilizan las verificaciones contenidas el documento VulnXML. Normalmente, esto podría significar varias cosas: que se perdieron paquetes, que hubo un problema de corte en algún enlace de red, o como efectivamente sucedió, que el servidor cortó la conexión, a causa de la existencia de la vulnerabilidad atacada que permite cambiar la secuencia de ejecución lógica de la aplicación a través de un desbordamiento de buffer.

En síntesis, de las doce capturas de paquetes procesadas, Snort generó un total de 13 alertas (recordemos que uno de los ataques exitosos generó dos alertas distintos). Para estos mismos ataques, el prototipo indicó que 3 capturas contenían ataques exitosos, 2 contenían ataques con resultado desconocido, y 7 contenían ataques fallidos, como se muestra en las figuras 4.12 y 4.13. De esto podemos concluir que, en estas primeras pruebas, el prototipo permite restar importancia al 58 % de los ataques.

Figura 4.12: Resultados del Prototipo

	ataque IIS +.htr	ataque IIS Unicode	ataque IIS ISAPI .printer
Equipo 1	success	success	unknown
Equipo 2	failure	failure	failure
Equipo 3	failure	failure	unknown

Figura 4.13: Resultados del Prototipo

	ataque php-ping
Equipo 2 - php-ping vulnerable	success
Equipo 2 - php-ping corregido	failure
Equipo 2 - sin php-ping	failure

Este resultado coincide con el objetivo principal de este trabajo, que buscaba reducir la cantidad de falsas alarmas.

Capítulo 5

Análisis de resultados

5.1. Análisis de tráfico real

Una vez realizadas las pruebas del prototipo en el laboratorio, se utilizó el mismo en un ámbito más cercano al comportamiento real del tráfico de una red. Para esto, se decidió analizar el tráfico de red generado durante el evento "Capture the Flag" 2001. Dicho evento se realizó en el marco de la conferencia anual *DefCon*[1], que reune a un importante número de expertos en seguridad informática. El evento consiste en una competencia entre distintos equipos. El ganador es quien logra mantener seguros los servicios de servidores preinstalados provistos por la organización del evento. Cada equipo debe proteger sus equipos y atacar a los servidores de los demás.

El evento dura tres días y posteriormente se publican las capturas del tráfico de red para poder realizar tareas de investigación. Estas capturas son de particular interés para la comunidad relacionada con los Sistemas de Detección de Intrusiones.

Las capturas de tráfico fueron obtenidas a través del grupo "The Smoo Group´´en `http://www.shmoo.com/cctf/`. Este tráfico es probablemente más virulento que el tráfico normal de una red conectada directamente a Internet, pero muy efectivo para encontrar gran variedad y cantidad de ataques. Se eligió esta captura en lugar de utilizar tráfico generado en forma artificial, debido a que utilizar tráfico sintético es generalmente considerado poco útil para realizar pruebas de efectividad de un IDS[35].

La captura mencionada se realizó en formato pcap[2]. Debido a que este trabajo sólo trata tráfico relacionado con el protocolo HTTP, se filtró la gran cantidad de paquetes sin relación con el mismo. Se utilizaron las herramientas tcpdump (`http://www.tcpdump.org`) y mergecap (`http://www.ethereal.com`) para obtener un nuevo archivo en formato

[1] `http://www.defcon.org`
[2] `http://www.tcpdump.org`

pcap que contiene únicamente las sesiones dirigidas al puerto 80/tcp, puerto utilizado comúnmente para HTTP.

5.2. Ejecución de Snort

Una vez obtenidos todos los paquetes correspondientes a sesiones TCP al puerto 80, se utilizó el IDS Snort para determinar las alertas generadas por el mismo.

Se configuró el programa Snort versión 2.0.0 para que reensamble los streams TCP,tanto para el cliente como para el servidor. Se eliminaron los preprocesadores irrelevantes (por ejemplo, los que inspeccionan tráfico referido a RPC) para mejorar la performance del programa. Esta ejecución de Snort resultó en los datos de la figura 5.1. Se encontraron más de cuatro millones de paquetes TCP, que correspondían a 1685621 conexiones. Analizando ese tráfico, Snort generó 100243 alertas, correspondientes a 420 reglas diferentes.

Figura 5.1: Paquetes capturados

paquetes tcp	4088326
streams tcp	1685621
alertas	100243
alertas diferentes	420

Al revisar las alertas generadas, se detectó que 72134 correspondían a *búsquedas automáticas de puertos de red activos*, o *port scannings*, realizadas con la herramienta nmap[3]. Por no corresponder a tráfico del protocolo HTTP, fueron eliminadas de la muestra.

Una vez restados las 72134 alertas relacionados con la búsqueda de puertos de red activos, se obtuvieron los números de la figura 5.2. En el cuadro se incluyen las categorías a las que corresponde cada regla según la taxonomía definida por Snort.

Como puede observarse en los datos obtenidos, Snort generó un número elevado de alertas. Esta cantidad es realmente excesiva, dificulta enormemente la revisión de las mismas por parte del operador y, por ende, se torna imposible tomar las medidas necesarias ante los ataques realmente peligrosos.

[3] `http://www.insecure.org`

Figura 5.2: Alertas

Alertas	28109
Alertas diferentes	419
Categorías de alerta	8
web-application-activity	6434
web-application-attack	12929
attempted-recon	6310
attempted-user	62
attempted-admin	78
attempted-dos	3
bad-unknown	454
unclassified	1839

5.3. Generación de documentos VulnIdsXML

Para poder utilizar el prototipo y comparar los resultados del mismo contra los resultados generados por el Snort, es necesario describir los ataques con el lenguaje VulnIdsXML. Como el número de alertas distintas detectadas por Snort resultó muy elevado, se decidió elegir un subconjunto reducido de las vulnerabilidades encontradas para ejecutar el prototipo.

Con el objetivo de asegurar que esta selección resulte representativa de la muestra original, se agruparon las alertas por vulnerabilidad (regla de Snort), y se ordenaron por número de apariciones. Posteriormente, se eligieron 21 vulnerabilidades (un 5 % de la muestra original), tomadas equidistantemente entre las 419 vulnerabilidades ordenadas según lo descripto anteriormente.

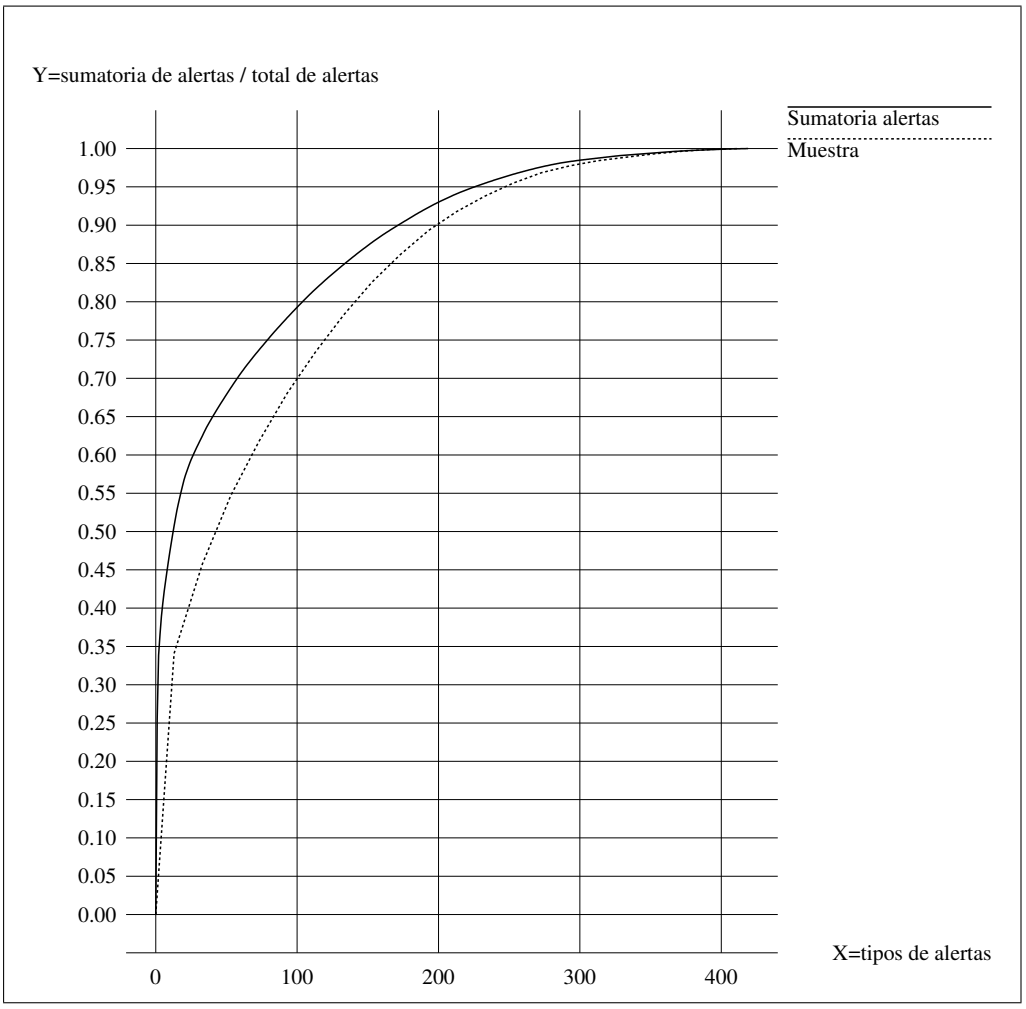
En la figura 5.3 se puede ver graficado, con línea continua:

$$y = \sum_{i=0}^{x} \frac{\#alertas_i}{total\_alertas}$$

Donde $\#alertas_i$ representa la cantidad de alertas de la clase i, y el denominador $total\_alertas$ corresponde al total de alertas detectadas y es igual a 28109.

En la misma figura, con línea punteada, se puede ver la misma relación, pero sólo con las 21 vulnerabilidades elegidas. En este caso el denomindor contabiliza únicamente las alertas correspondientes a la muestra, y resultando un total de 889 alertas. La inclusión del denominador $total\_alertas$ nos permite normalizar las curvas para su comparación. La similitud de las curvas muestra la similitud de la distribución del subconjunto elegido con el universo original.

Figura 5.3: Sumatoria de las alertas detectadas por Snort



En la figura 5.4 se listan las reglas elegidas para correr el prototipo, junto con la cantidad de alertas que generó el Snort para cada una, sumando un total de 889 alertas.

La generación de los documentos VulnIdsXML correspondientes a las 21 reglas de Snort seleccionadas no resultó una tarea trivial, ya que fue necesario investigar cada vulnerabilidad y documentar todo la información incluida en VulnIdsXML, tanto en lo que corresponde a la descripción informal como a la forma de identificar los comportamientos de los servidores, ya sean estos vulnerables o no. En el futuro, esta tarea puede verse facilitada si el VulnXML se convierte en un lenguaje aceptado y utilizado por la comunidad, ya que soló habría que agregar la extensión correspondiente a VulnIdsXML a los documentos VulnXML generados por la comunidad.

Figura 5.4: Muestra de Vulnerabilidades

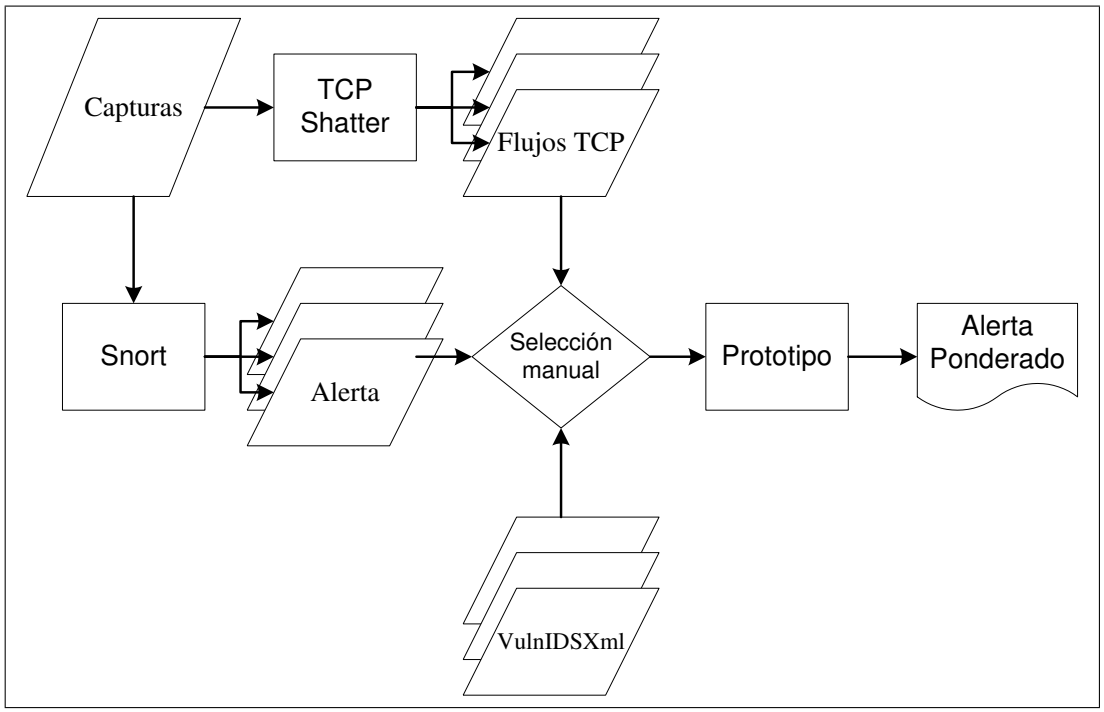
#alertas	Snort ID	Descripción
303	1122	WEB-MISC /etc/passwd
104	1042	WEB-IIS view source via translate header
79	810	WEB-CGI whois_raw.cgi access
63	1482	WEB-CGI view_source access
56	901	WEB-CGI webspirs.cgi access
48	1570	WEB-CGI loadpage.cgi access
43	1157	WEB-MISC netscape PublishingXpert 2 Exploit
38	1650	WEB-CGI tst.bat access
32	1513	WEB-CGI input.bat access
28	883	WEB-CGI flexform access
22	1765	WEB-CGI Nortel Contivity cgiproc access
18	1129	WEB-MISC .htaccess access
15	1134	WEB-MISC Phorum admin access
12	964	WEB-FRONTPAGE users.pwd access
8	1075	WEB-MISC postinfo.asp access
6	1521	WEB-MISC server-status access
4	1571	WEB-CGI dcforum.cgi directory traversal attempt
4	930	WEB-COLDFUSION snippets attempt
3	1187	WEB-MISC SalesLogix Eviewer web command attempt
2	957	WEB-FRONTPAGE registrations.txt access
1	1542	WEB-CGI cgimail access
889	**Total**	

5.4. Ejecución del prototipo

Para evaluar la ejecución de Snort con la muestra elegida, se utilizó un único archivo de datos que contenía un conjunto de sesiones HTTP. Sin embargo, el prototipo desarrollado en este trabajo procesa archivos en formato pcap que contengan una única sesión HTTP. Para adecuar la captura a los requerimientos del prototipo se utilizó una versión modificada de la herramienta *tcpshatter* (ver Anexo D), que toma como entrada el nombre de un archivo que contiene n sesiones, en formato PCAP, y lo separa en n archivos, conteniendo una única sesión cada uno. Debido a la gran cantidad de sesiones de la captura original (1.685.621), debimos realizar pequeñas modificaciones en el programa *tcpshatter* para que los archivos se generen dentro de diversos directorios, y así facilitar el acceso al filesystem.

Una vez separadas las sesiones, se identificaron, utilizando pequeños scripts ad hoc, las sesiones correspondientes a cada alerta de Snort, y se ejecutó el prototipo para cada una de ellas. Un esquema completo de este proceso puede verse en la figura 5.5.

Figura 5.5: Flujo de una detección con el prototipo



5.5. Análisis de resultados

En esta sección se comparan los resultados obtenidos originalmente por el Snort, con los obtenidos por el prototipo. Como vimos en la figura 5.4, Snort genero 889 alertas, sin distinguir si los ataques fueron exitosos o no.

En la figura 5.6 se puede ver la clasificación de las alertas obtenida por el prototipo:

Figura 5.6: Resultados del Prototipo

success	2
success (falso positivo)	18
failure	731
unknown	132
error	6
total	889

De estos resultados se desprende que se pueden descartar con seguridad 731 alertas (las que resultaron en la categoría *failure*), es decir el 82 %, debido a que se puede afirmar

que el ataque no fue exitoso. En su mayoría, los ataques requerían el envío de un recurso específico que no se encontraba en el servidor, por lo cual este respondió con el código de error *404 (Not Found)*.

De las restantes alertas, en 132 (un 14 %) no se puede identificar con seguridad cuál fue el resultado. Algunas de estas alertas corresponden a eventos en los que no se obtuvo una respuesta del servidor que pudiese indicar el éxito o la falla del intento de ataque. En otros casos, y debido a la escasez de información pública acerca de las vulnerabilidades relacionadas, no se pudo definir concretamente en el documento VulnIdsXML en que casos el ataque era exitoso, y entonces sólo se pudo descartar los casos en los que se estaba seguro de que el ataque fue fallido.

Finalmente, 20 alertas indicaron un ataque exitoso. De las mismas, 2 alertas indicaron casos reales de ataques, ambos referidos a la obtención ilegítima de un archivo *.asa* de un servidor *Internet Information Server*. La vulnerabilidad involucrada fue la descripta por la regla de Snort *1042* "WEB-IIS view source via translate header".

Las restantes 18 alertas corresponden a falsos positivos, todos generados por un servidor configurado para devolver respuestas no estándar al solicitar un recurso inexistente (Devolvió un código de error *200* junto a un mensaje de bienvenida). Estos falsos positivos se podrían erradicar implementando una idea surgida en las discusiones del grupo promotor de WasXML. La idea es simple, y consiste en determinar a priori, mediante la generación por parte del IDS de solicitudes HTTP, cual es la respuesta de un servidor cuando se le solicita un recurso inexistente. Al conocer las propiedades de esta respuesta, se puede determinar con seguridad en que casos el servidor indicó que el recurso no existe, aunque el servidor no cumpla con los estándares. En este trabajo no se implementó la idea porque no estamos trabajando con los servidores en tiempo real.

Además, el prototipo no pudo tratar correctamente 6 casos, debido a falencias en la heurística de seguimiento de sesiones TCP del prototipo.

Considerando que las alertas clasificadas como *failure* por el prototipo pueden ser ignoradas por un operador, sólo se requerirá atención sobre un 18 % de las alertas identificadas originalmente por Snort. El porcentaje de alertas a revisar en forma manual se puede reducir aún más mejorando los documentos VulnIdsXML, y utilizando otras técnicas complementarias para detectar falsas alarmas, como por ejemplo la utilización de técnicas de identificación automática de servidor[36, 37].

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este trabajo se analizó el funcionamiento de los Sistemas de Detección de Intrusiones, y se propuso un modelo que reduce considerablemente la cantidad de falsas alarmas. El modelo se basa en la definición de un lenguaje que permite describir los posibles ataques a servidores y aplicaciones web, teniendo en cuenta las respuestas del servidor y las características del protocolo HTTP, además de las solicitudes de los clientes como se realiza habitualmente.

Se desarrolló un prototipo que implementa el modelo, con el objetivo de demostrar su eficacia. Se realizaron pruebas en un laboratorio instalado ad-hoc, realizando ataques específicos sobre vulnerabilidades existentes en los servidores. Posteriormente, se realizaron pruebas utilizando capturas de tráfico real. Las pruebas se efectuaron utilizando el IDS Snort, uno de los productos más utilizados en el mercado, y el prototipo, y se presentó un exhaustivo análisis comparativo de los resultados de ambos.

En el análisis se pudo comprobar empíricamente la validez del modelo. En las distintas pruebas se obtuvieron resultados en los que el prototipo descartaba más del 80 % de las alertas generadas por Snort que correspondían a falsas alarmas. Esta reducción en la cantidad de alertas facilita la tarea del administrador de IDS, y permite elevar el grado de utilidad real de estos sistemas y la factibilidad de su manejo.

La realización de este trabajo se vio dificultada por la escasez de investigaciones formales sobre temas similares, y el dinamismo de las tecnologías de detección de intrusiones[5].

El prototipo desarrollado queda a disposición del público. Esta herramienta permite continuar con las tareas de análisis de resultados en base a cualquier tráfico HTTP.

6.2. Trabajo Futuro

Es posible extender las ideas presentadas en este trabajo en diferentes formas. A continuación se presentan las líneas de acción que consideramos más interesantes.

- Extender las ideas propuestas a otros protocolos y, posteriormente, generalizar el concepto. Si bien en este trabajo se utilizó el protocolo HTTP, se podrían desarrollar trabajos similares para otros protocolos de aplicación como FTP o DNS, entre otros.
- Mejorar la implementación de esta idea incorporándola a un IDS de red completo que trabaje en tiempo real o en forma totalmente automatizada. Además podría extenderse incorporando técnicas de identificación automática de servidores web para ayudar a la decisión sobre el éxito del ataque.
- Extender el prototipo para que no sólo pondere el ataque, sino que además incluya información adicional, como por ejemplo si el web server corresponde con el elemento *applicable_to* definido en el VulnIdsXML, información adicional sobre la taxonomía de la vulnerabilidad, etc.
- Migrar la modificación propuesta del lenguaje VulnXML al lenguaje WAS XML, cuando el mismo se encuentre disponible. Este lenguaje incorpora además una taxonomía de ataques mejorada.

Referencias

[1] T. Belcher and Elad Yoran. The riptech internet security threat report. `http://www.riptech.com`, July 2002.

[2] D. Ahmad, C. L. Arnold, B. Dunphy, M. Prosser, and V. Weafer. Symantec Internet Security Threat Report. `http://www.symantec.com`, January 2003.

[3] Martin Roesch. SNORT - Lightweight intrusion detection for networks. In *LISA '99*. The USENIX Association, 1999.

[4] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[5] Julia Allen, Alan Christie, et al. State of the Practice of Intrusion Detection Technologies . Technical report, Software Engineering Institute, Carnegie Mellon Univeristy, 1999.

[6] Samuel Patton, William Yurcik, and David Doss. An Achilles'Heel in Signature-Based IDS: Squealing False Positives in SNORT. In *RAID 2001*. University of California-Davis, 2001.

[7] Thomas Ptacek and Timothy Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection, 1998.

[8] kc claffy. the nature of the beast: recent traffic measurements from an internet backbone, April 1998.

[9] Euclidian Consulting. Distributed Intrusion Detection System, Top 10 target ports, April 2003. `http://www.dshield.org/topports.html`.

[10] Teresa F. Lunt. Automated audit trail analysis and intrusion detection: A survey. In *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD, 1988.

[11] Incident Taxonomy and Description Working Group. Taxonomy of the Computer Security Incident related terminology . `http://www.terena.nl/tech/task-forces/tf-csirt/iodef/docs/i-taxonomy_terms.html`. Work in progress, 2002.

[12] D. Curry and H. Debar. Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition , January 2004. Internet Draft draft-ietf-idwg-idmef-xml-11.txt. Work in progress. Expires: July 8, 2004.

[13] Lance Spitzner. Definitions and value of honeypots, May 2003.

[14] R. Fielding, UC Irvine, et al. RFC2616 Hypertext Transfer Protocol – HTTP/1.1, June 1999.

[15] T. Berners-Lee, R. Fielding, UC Irvine, et al. RFC2396 Uniform Resource Identifiers (URI): Generic syntax, August 1998.

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

[17] Zheng Zhang, Jun Li, C.N. Manikopoulos, Jay Jorgenson, and Jose Ucles. Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY, June 2001.

[18] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. Nadir: an automated system for detecting network intrusion and misuse. *Comput. Secur.*, 12(3):235–248, 1993.

[19] Sandeep Kumar and Eugene H. Spafford. An application of pattern matching in intrusion detection. Technical report, The COAST Project Department of Computer Sciences Purdue University, June 1994.

[20] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communication Security (CCS '03)*, pages 251–261, Washington, DC, October 2003. ACM Press.

[21] G. Vigna, W. Robertson, V. Kher, and R.A. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.

[22] Jeremiah Grossman. Identifying Web Servers, A first-look into Web Server Fingerprinting. In *BlackHat Asia 2002*. Black Hat, Inc, 2002.

[23] Gonzalo Álvarez and Slobodan Petrovic. A new taxonomy of web attacks suitable for efficient encoding. *Computers & Security*, 22(5):435–449, April 2003.

[24] Frédéric Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *17th Annual Computer Security Applications Conference (ACSAC.01)*. IEEE, 2001.

[25] Sai Krishna V. Intrusion detection techniques: Pattern matching and protocol analysis. `http://www.giac.org/practical/GSEC/Sai_Krishna_GSEC.pdf`, 2003.

[26] Mark Handley and Vern Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX secuiryty symp.*, Washington, DC, August 2001.

[27] Daniel Lowry Lough. A Taxonomy of Computer Attacks with Applications to Wireless Networks. Master's thesis, Virginia Polytechnic Institute and State University, 2001.

[28] Marcus J. Ranum, Kent Landfield, Mike Stolarchuk, Mark Sienkiewicz, Andrew Lambeth, and Eric Wall. Implementing A generalized tool for network monitoring. In *Proceedings of the Eleventh Systems Administration Conference (LISA '97)*, San Diego, CA, 1997.

[29] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. In *Proceedings of the* 1^{st} *ACM Workshop on Intrusion Detection Systems*, Athens, Greece, November 2000.

[30] Mark Curphey and Jennifer Tharp. VulnXML Vision Document, July 2002.

[31] XML Core Working Group. Extensible markup language (xml). `http://www.w3.org/XML/`, 2003.

[32] Marcus J. Ranum. False positives: A user´s guide to making sense of ids alarms, February 2003.

[33] Mark Gerken. Statistical-based intrusion detection, 1997.

[34] Mark Norton and Daniel Roelker. Snort 2.0 protocol flow analyzer, 2003.

[35] M Ranum. Experiences benchmarking intrusion detection systems. Technical report, NFR Security, Inc, December 2001.

[36] Fyodor. Remote os detection via tcp/ip stack fingerprinting. `http://www.insecure.org/nmap/nmap-fingerprinting-article.html`, 1998.

[37] Julien Bordet. Remote smtp server detection. `http://www.greyhats.org/outils/smtpscan/remote_smtp_detect.pdf`, 2002.

Apéndice A

Ejemplo VulnXML

```
<!DOCTYPE WebApplicationTest SYSTEM "VulnXML-1.4-comments.dtd">
<WebApplicationTest>
  <TestDescription name="OWASP-00002" version="0.1" released="2002-04-10" updated="2002-04-30"
                   protocol="HTTP" affects="server" severity="high" alert="success"
                   type="Overflows" mayproxy="true">
    <Reference database="Bugtraq" URL="http://www.securityfocus.com/bid/4485"/>
    <Reference database="CVE"
               URL="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079"/>
    <Reference database="Microsoft"
               URL="http://www.microsoft.com/technet/security/bulletin/ms02-018.asp"/>
    <Reference database="Cert" URL="http://www.cert.org/advisories/CA-2002-09.html"/>
    <Copyright>Public Domain</Copyright>
    <Description>Buffer overflow in the chunked encoding transfer mechanism in Internet
                 Information Server (IIS) 4.0 and 5.0 Active Server Pages allows attackers to
                 cause a denial of service or execute arbitrary code.</Description>
    <ApplicableTo>
      <Platform>
        <OS>Windows</OS>
        <Arch>i386</Arch>
      </Platform>
      <WebServer>Microsoft-IIS</WebServer>
      <WebServer>Mivrosoft-IIS/[45].*</WebServer>
    </ApplicableTo>
    <TriggerOn event="file">
      <Match type="regex">.*\.asp</Match>
    </TriggerOn>
    <Impact>The attacker can cause the web server to crash and restart, and could potentially
            execute arbitrary code on the web server</Impact>
    <Recommendation>Delete sample ASP scripts to deter bulk scanners. Install the patch
                    supplied by Microsoft as soon as it is available.</Recommendation>
  </TestDescription>
  <Connection scheme="http://" host="" port="80">
    <Step name="step1">
      <Request>
        <MessageHeader>
          <Method encoding="text">POST</Method>
          <URI encoding="text">${scheme}://${host}:${port}/${path}/${file}</URI>
          <Version encoding="text">HTTP/1.1</Version>
          <Header>
            <Name>Accept</Name>
            <Value>*/*</Value>
          </Header>
          <Header>
```

```
      <Name>Host</Name>
      <Value>${host}</Value>
    </Header>
    <Header>
      <Name>Transfer-Encoding</Name>
      <Value>chunked</Value>
    </Header>
    <Header>
      <Name>Content-Type</Name>
      <Value>application/x-www-form-urlencoded</Value>
    </Header>
    <Header>
      <Name>Content-Length</Name>
      <Value>length=auto</Value>
    </Header>
  </MessageHeader>
  <MessageBody>
    <Separator encoding="text"/>
    <Item encoding="base64">MQpFCjAKCgoK</Item>
  </MessageBody>
</Request>
<Response>
  <SetVariable name="ResponseCode" type="string">
    <Description>HTTP Response code</Description>
    <Source source="status">^.*\s(\d\d\d)\s</Source>
  </SetVariable>
  <SetVariable name="redir302" type="string">
    <Description>See if we got a custom 404 handler, correctly implemented using a
                 redirection</Description>
    <Source source="header">Location: (.*)$</Source>
  </SetVariable>
  <SetVariable name="body404" type="string">
    <Description>See if we got a custom 404 handler, incorrectly implemented using a
                 return code of 200</Description>
    <Source source="body">(404 Not Found)</Source>
  </SetVariable>
  <SetVariable name="unpatched" type="string">
    <Description>An unpatched server returns "(0x80004005)&lt;br&gt;Unspecified</Description>
    <Source source="body">(\(0x80004005\)&lt;br&gt;Unspecified)</Source>
  </SetVariable>
  <SetVariable name="patched" type="string">
    <Description>A patched server returns "(0x80004005)&lt;br&gt;Request</Description>
    <Source source="body">(\(0x80004005\)&lt;br&gt;Request)</Source>
  </SetVariable>
</Response>
<TestCriteria type="failure">
  <ErrorMessage>The page was not found</ErrorMessage>
  <Compare variable="${ResponseCode}" test="eq">
    <Value>200</Value>
    <Compare variable="${body404}" test="neq">
      <Value>404</Value>
    </Compare>
  </Compare>
  <Compare variable="${ResponseCode}" test="eq">
    <Value>404</Value>
  </Compare>
  <Compare variable="${ResponseCode}" test="eq">
    <Value>302</Value>
  </Compare>
  <Compare variable="${ResponseCode}" test="eq">
    <Value>500</Value>
  </Compare>
</TestCriteria>
<TestCriteria type="failure">
  <Compare variable="${patched}" test="neq">
```

```
            <Value/>
          </Compare>
        </TestCriteria>
        <TestCriteria type="success">
          <Compare variable="${unpatched}" test="neq">
            <Value/>
          </Compare>
        </TestCriteria>
      </Step>
    </Connection>
  </WebApplicationTest>
```

Apéndice B

DTD VulnIdsXML: fragmento agregado

```
<!ELEMENT Connection (Step+)>
<!ATTLIST Connection
  scheme CDATA #REQUIRED
  host CDATA #REQUIRED
  port CDATA #REQUIRED
>

<!-- Aquí se agregó el elemento IDSDetectRequest para diferenciarlo del patrón de detección -->
<!ELEMENT Step (IDSDetectRequest?, Request?, Response, TestCriteria+)>
<!ATTLIST Step
  name CDATA #REQUIRED
>

<!-- Esta definición de request es utilizada para  detectar ataques, a diferencia de la original
     que se utiliza  para generar un test de vulnerabilidad.-->
<!ELEMENT IDSDetectRequest (MessageHeader?, MessageBody?)>
<!-- Request describes the request string that is sent to the web server. One
may want to have no header if one is continuing a "Transfer-Encoding: chunked"
message, for instance, and simply want to transfer data. One may have no
MessageBody when doing a HEAD or GET request, for instance.-->
<!ELEMENT Request (MessageHeader?, MessageBody?)>
```

Apéndice C

Documentos VulnIdsXML

A continuación se presentan los cuatro documentos VulnIdsXML desarrollados para las primeras pruebas del prototipo en el laboratorio.

C.1. ASAhtr.xml - Vulnerabilidad relacionada con extensión .htr

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--$Id: ASAHtr.xml,v 1.6 2004/07/05 23:17:31 tomas Exp $-->
<!DOCTYPE WebApplicationTest SYSTEM "VulnIdsXML-1.0-comments.dtd">
<WebApplicationTest>
  <TestDescription name="OWASP-90002" mayproxy="true" affects="file" severity="high" alert="success"
       protocol="HTTP" updated="2003-09-11" released="2003-09-11" version="0.0" type="Validation">
    <Reference database="Snort" URL="http://www.snort.org/snort-db/sid.html?sid=1725"/>
    <Reference database="CVE" URL="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0630"/>
    <Copyright>Public Domain</Copyright>
    <Description>WEB-IIS +.htr code fragment attempt</Description>
    <ApplicableTo>
      <Platform>
        <OS>Windows</OS>
        <Arch>i386</Arch>
      </Platform>
      <WebServer>Microsoft-IIS</WebServer>
      <WebServer>Microsoft-IIS[45]</WebServer>
      <ApplicationServer/>
    </ApplicableTo>
    <TriggerOn event="file">
      <Match type="regex">.*</Match>
    </TriggerOn>
    <Impact>Information gathering.  Fragments of the source code of an ASP may
            be returned possibly disclosing sensitive information.
    </Impact>
    <Recommendation>Apply the patch referenced in the Microsoft link.
                    Consider running the IIS Lockdown Tool to disable HTR
                    functionality.
    </Recommendation>
  </TestDescription>
  <Connection scheme="http://" host=".*" port="80">
    <Step name="step1">
```

```
      <IDSDetectRequest>
        <MessageHeader>
          <Method>.*</Method>
          <URI encoding="text">\+\.htr</URI>
          <Version>.*</Version>
        </MessageHeader>
      </IDSDetectRequest>
      <Response>
        <SetVariable name="ResponseCode" type="string">
          <Description>HTTP Response code</Description>
          <Source source="status">^.*\s(\d\d\d)\s</Source>
        </SetVariable>
      </Response>
      <TestCriteria type="success">
        <ErrorMessage>Request Successfull</ErrorMessage>
        <Compare variable="${ResponseCode}" test="eq">
          <Value>200</Value>
        </Compare>
      </TestCriteria>
      <TestCriteria type="failure">
        <ErrorMessage>request unsuccessful</ErrorMessage>
        <Compare variable="${ResponseCode}" test="neq">
          <Value>200</Value>
        </Compare>
      </TestCriteria>
    </Step>
  </Connection>
</WebApplicationTest>
```

C.2. execIIS.xml - Vulnerabilidad relacionada con codificación del caracter "/"

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--$Id: execIIS.xml,v 1.6 2004/07/05 23:17:31 tomas Exp $-->
<!DOCTYPE WebApplicationTest SYSTEM "VulnIdsXML-1.0-comments.dtd">
<WebApplicationTest>
  <TestDescription name="OWASP-90003" version="0.0" released="2003-09-11" protocol="HTTP"
      affects="file" updated="2003-09-11" severity="high" alert="success" mayproxy="true"
       type="Canonicalisation">
    <Reference database="Snort" URL="http://www.snort.org/snort-db/sid.html?sid=1945"/>
    <Reference database="CVE" URL="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884"/>
    <Copyright>Public Domain</Copyright>
    <Description>WEB-IIS unicode directory traversal attempt</Description>
    <ApplicableTo>
      <Platform>
        <OS>Windows</OS>
        <Arch>i386</Arch>
      </Platform>
      <WebServer>Microsoft-IIS</WebServer>
      <WebServer>Microsoft-IIS[345]</WebServer>
      <ApplicationServer/>
    </ApplicableTo>
    <TriggerOn event="file">
      <Match type="regex">.*</Match>
    </TriggerOn>
    <Impact>Remote access.  This attack can allow an attacker to execute commands a
           vulnerable IIS server.
    </Impact>
    <Recommendation>Apply the patch referenced in the Microsoft link.
    </Recommendation>
```

```
  </TestDescription>
  <Connection scheme="http://" host=".*" port="80">
    <Step name="step1">
      <IDSDetectRequest>
        <MessageHeader>
          <Method>.*</Method>
          <URI encoding="text">/..%255c..</URI>
          <Version>.*</Version>
        </MessageHeader>
      </IDSDetectRequest>
      <Response>
        <SetVariable name="ResponseCode" type="string">
          <Description>HTTP Response code</Description>
          <Source source="status">^.*\s(\d\d\d)\s</Source>
        </SetVariable>
      </Response>
      <TestCriteria type="success">
        <ErrorMessage>Reques Successfull</ErrorMessage>
        <Compare variable="${ResponseCode}" test="eq">
          <Value>200</Value>
        </Compare>
      </TestCriteria>
      <TestCriteria type="failure">
        <ErrorMessage>request unsuccessful</ErrorMessage>
        <Compare variable="${ResponseCode}" test="neq">
          <Value>200</Value>
        </Compare>
      </TestCriteria>
    </Step>
  </Connection>
</WebApplicationTest>
```

C.3. jill.xml - Vulnerabilidad relacionada con extensión Internet Printing ISAPI

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--$Id: jill.xml,v 1.7 2004/07/05 23:17:31 tomas Exp $-->
<!DOCTYPE WebApplicationTest SYSTEM "VulnIdsXML-1.0-comments.dtd">
<WebApplicationTest>
  <TestDescription name="OWASP-90004" version="0.0" released="2003-09-11" updated="2003-09-11"
      mayproxy="true" protocol="HTTP" affects="file" type="Overflows" alert="success"
      severity="high">
    <Reference database="Snort" URL="http://www.snort.org/snort-db/sid.html?sid=971"/>
    <Reference database="CVE" URL="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0241"/>
    <Reference database="arachnids" URL="http://www.whitehats.com/info/IDS533"/>
    <Copyright>Public Domain</Copyright>
    <Description>WEB-IIS ISAPI .printer access</Description>
    <ApplicableTo>
      <Platform>
        <OS>Windows</OS>
        <Arch>i386</Arch>
      </Platform>
      <WebServer>Microsoft-IIS</WebServer>
      <WebServer>Microsoft-IIS5</WebServer>
      <ApplicationServer/>
    </ApplicableTo>
    <TriggerOn event="file">
      <Match type="regex">.*</Match>
    </TriggerOn>
    <Impact>Possible system compromise.  The ".printer" vulnerability may result in
```

```
            remote system level access to an IIS 5.0 server.
    </Impact>
    <Recommendation>Install latest patches from the vendor, or disable the ".printer"
                    extensions using the IIS administration tool.
    </Recommendation>
  </TestDescription>
  <Connection scheme="http://" host=".*" port="80">
    <Step name="step1">
      <IDSDetectRequest>
        <MessageHeader>
          <Method>.*</Method>
          <URI encoding="text">.printer</URI>
          <Version>.*</Version>
        </MessageHeader>
      </IDSDetectRequest>
      <Response>
        <SetVariable name="ResponseCode" type="string">
          <Description>HTTP Response code</Description>
          <Source source="status">^.*\s(\d\d\d)\s</Source>
        </SetVariable>
      </Response>
      <TestCriteria type="failure">
        <ErrorMessage>request unsuccessful</ErrorMessage>
        <Compare variable="${ResponseCode}" test="eq">
          <Value>4..</Value>
        </Compare>
      </TestCriteria>
    </Step>
  </Connection>
</WebApplicationTest>
```

C.4. idCommandAttempt.xml - Vulnerabilidad relacionada con ejecución de comando “id” de Unix

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--$Id: idCommandAttempt.xml,v 1.7 2004/07/05 23:17:31 tomas Exp $-->
<!DOCTYPE WebApplicationTest SYSTEM "VulnIdsXML-1.0-comments.dtd">
<WebApplicationTest>
  <TestDescription name="OWASP-90001" version="0.0" released="2003-09-04" updated="2003-09-04"
      protocol="HTTP" mayproxy="true" affects="file" alert="success" severity="high"
      type="ParameterManipulation">
    <Reference database="Snort" URL="http://www.snort.org/snort-db/sid.html?sid=1333"/>
    <Copyright>Public Domain</Copyright>
    <Description>Attempted id command access via web</Description>
    <ApplicableTo>
      <Platform>
        <OS>Unix</OS>
        <Arch>i386</Arch>
      </Platform>
      <WebServer/>
    </ApplicableTo>
    <TriggerOn event="file">
      <Match type="regex">.*</Match>
    </TriggerOn>
    <Impact>Attempt to gain information on users and groups that exist on the
            host using the id command.</Impact>
    <Recommendation>Webservers should not be allowed to view or execute files
                    and binaries outside of it’s designated web root or cgi-bin.
    </Recommendation>
  </TestDescription>
```

```
  <Connection scheme="http://" host=".*" port="80">
    <Step name="step1">
      <IDSDetectRequest>
        <MessageHeader>
          <Method>.*</Method>
          <URI encoding="text">;id</URI>
          <Version>.*</Version>
        </MessageHeader>
      </IDSDetectRequest>
      <Response>
        <SetVariable name="ResponseCode" type="string">
          <Description>HTTP Response code</Description>
          <Source source="status">^.*\s(\d\d\d)\s</Source>
        </SetVariable>
        <SetVariable name="idmessage" type="string">
          <Description>A vulnerable server returns "uid=[0-9]+\(0-9a-zA-Z-]+\)"
          </Description>
          <Source source="body">(uid=[0-9]+\(0-9a-zA-Z-]+\))</Source>
        </SetVariable>
      </Response>
      <TestCriteria type="success">
        <ErrorMessage>id message found</ErrorMessage>
        <Compare variable="${idmessage}" test="neq">
          <Value/>
          <Compare variable="${ResponseCode}" test="eq">
            <Value>200</Value>
          </Compare>
        </Compare>
      </TestCriteria>
      <TestCriteria type="failure">
        <ErrorMessage>request unsuccessful</ErrorMessage>
        <Compare variable="${idmessage}" test="neq">
          <Value/>
          <Compare variable="${ResponseCode}" test="neq">
            <Value>200</Value>
          </Compare>
        </Compare>
      </TestCriteria>
      <TestCriteria type="failure">
        <ErrorMessage>id message not seen</ErrorMessage>
        <Compare variable="${idmessage}" test="eq">
          <Value/>
        </Compare>
      </TestCriteria>
    </Step>
  </Connection>
</WebApplicationTest>
```

Apéndice D

Tcp-shatter

La herramienta Tcp-shatter fue desarrollada por John Bashinski, en la Ohio State University, como herramienta auxiliar para otras herramientas de manejo de capturas de red. Esta herramienta genera un archivo para cada sesión TCP que encuentre en el archivo de entrada. Debido a la gran cantidad de sesiones de la captura original (1.685.621), debimos realizar pequeñas modificaciones para que los archivos se generen dentro de diversos directorios, y así facilitar el acceso al filesystem. A continuación, incluimos el archivo con las modificaciones en formato diff.

```
diff -ru tcpshatter/tcpshatter.c tcpshatter-new/tcpshatter.c
--- tcpshatter/tcpshatter.c        1998-12-08 14:13:41.000000000 -0300
+++ tcpshatter-new/tcpshatter.c        2004-03-08 16:43:33.000000000 -0300
@@ -57,10 +57,12 @@
 #include <netinet/ip_icmp.h>
 #include <netinet/tcp.h>
 #include <netinet/udp.h>
+#include <dirent.h>
+#include <sys/stat.h>

 /** Configuration constants
  **/
-#define CACHESIZE 5                     /* Number of files to keep open */
+#define CACHESIZE 50                     /* Number of files to keep open */

 /** Internal structures
  **/
@@ -280,7 +282,9 @@
   u_int32_t highip;
   u_int16_t highport;
   pcap_dumper_t *outstream;
-
+  DIR *savedir;
+  u_int32_t dirl, dirh;
+
   if (ipa < ipb) {
     lowip = ipa;
     lowport = porta;
@@ -303,13 +307,38 @@
     highport = porta;
   }
```

```
+  dirl = lowip >> 24 ;
+  dirh = highip >> 24 ;
   nameptr = namebuf;
+  /* acá creamos el directorio (si no existe) */
+
+  sprintf ( nameptr, "%d", dirl);
+  savedir = opendir(namebuf);
+  if (! savedir)
+        mkdir(namebuf,S_IRWXU);
+  else
+          closedir(savedir);
+
+  while (*nameptr != '\0') nameptr++;
+
+  nameptr = strappend (nameptr, "/", namebuf + sizeof(namebuf));
+  sprintf ( nameptr, "%d", dirh);
+  savedir = opendir(namebuf);
+  if (! savedir)
+        mkdir(namebuf,S_IRWXU);
+  else
+          closedir(savedir);
+
+  while (*nameptr != '\0') nameptr++;
+  nameptr = strappend (nameptr, "/", namebuf + sizeof(namebuf));
+
   nameptr = strappend (nameptr, opts->fileprefix, namebuf + sizeof(namebuf));
   nameptr = strappend (nameptr, filebase, namebuf + sizeof(namebuf));
   nameptr = strappend (nameptr, ":", namebuf + sizeof(namebuf));
   nameptr = inet_neta (ntohl(lowip), nameptr,
                        (namebuf + sizeof (namebuf) - nameptr));
   while (*nameptr != '\0') nameptr++;
+
   nameptr = strappend (nameptr, ".", namebuf + sizeof(namebuf));
   if (nameptr != NULL) {
     nameptr += snprintf (nameptr, (namebuf+sizeof(namebuf)) - nameptr,
@@ -325,6 +354,7 @@
                          "%u", highport);
   }

+  printf ("%s\n",namebuf);
   if ((outstream = getfilestream(namebuf, opts)) == NULL) {
     perror ("can't open file");
     exit(1);
```

Apéndice E

Contenido del CD-ROM

Con este trabajo se incluye un CD-ROM que contiene:

prototipo.pl: Código fuente del prototipo desarrollado para validar el modelo presentado en este trabajo. Utiliza Perl y las librerías Net::Pcap, NetPacket XML::LibXML y URI::Escape.

parse_xml.pl: Utilitario para visualizar la información descriptiva de la vulnerabilidad incluida en un documento VulnIdsXML

VulnXML-1.4-comments.dtd: DTD de VulnXML, lenguaje de descripción de vulnerabilidades en servidores y aplicaciones web.

VulnIdsXML-1.0-comments.dtd: DTD de VulnIdsXML, extensión de VulnXML propuesta en este trabajo.

vulnidsxml/*: Documentos VulnIdsXML correspondientes a los ataques utilizados, tanto en el laboratorio como en las pruebas de tráfico real.

capturas/*: Capturas de red que contienen los 889 ataques extraidos del evento "Capture The Flag".