ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"IMPLEMENTACIÓN DE UN WEB APPLICATION FIREWALL BASADO EN MOD_SECURITY"

TESINA DE SEMINARIO

Previa a la obtención de los títulos de

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

INGENIERO EN CIENCIAS COMPUTACIONALES ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

Presentada por

GEORGES BENJAMIN VICENTE FLAMENT JORDÁN

JOHANNA LISBETH VERA GUERRERO

GUAYAQUIL - ECUADOR

2011

AGRADECIMIENTO

A mi familia, mis amigos, y todos aquellos
maestros que supieron transmitir, más
allá de las palabras, una vocación
Benjamin

A mis padres, esposo, hermanas, y abuela,
por su apoyo incondicional por darme fuerzas
para alcanzar este objetivo,
a mi compañero de tesis por su comprensión y
paciencia durante estos meses de trabajo
Johanna

DEDICATORIA

A Peluso el gato feliz, y Vaka el gato patán

Benjamin

A mi hija

Johanna

TRIBUNAL DE SUSTENTACIÓN

Ing. Alfonso Aranda
Profesor del Seminario de Graduación

Ing. Albert Espinal
Profesor Delegado del Decano

Escaneado con CamScanner

DECLARACIÓN EXPRESA

La responsabilidad del contenido de esta Tesina de Seminario, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral

Benjamin Flament

Johanna Vera

Escaneado con CamScanner

RESUMEN

La necesidad de un dispositivo capaz de asegurar servidores web contra ataques de usuarios maliciosos es grande en la época en que vivimos, pues la mayoría de intentos de acceso no autorizado a sistemas se da justamente por sus aplicaciones web. Se vuelve entonces, no solo importante, sino imperativo el tener una capa extra de seguridad completamente especializada para proteger nuestras aplicaciones web.

El mercado actual cuenta con una gama de soluciones existentes pero de alto costo. El presente trabajo espera demostrar que es posible implementar un Web Application Firewall de bajo costo en relación a las alternativas existentes, de manera que tecnologías como estas se vuelvan accesibles por todos. Para lograr nuestro objetivo, el proyecto está basado completamente en software libre. El producto desarrollado tendrá, dentro de lo posible, las mismas funcionalidades que los productos existentes, de modo que pueda competir con ellos en eficacia.

El producto final será un Appliance completo que incluya hardware, software e interfaz de administración en un solo paquete utilizable por cualquier administrador de redes.

CONTENIDO

AGRADECIMIENTO i

DEDICATORIA ii

TRIBUNAL DE SUSTENTACIÓN iii

DECLARACIÓN EXPRESA iv

RESUMEN v

CONTENIDO vi

ABREVIATURAS ix

INDICE DE GRÁFICOS xi

CAPITULO 1

ANTECEDENTES Y JUSTIFICACIÓN 1

1.1 DESCRIPCIÓN DEL PROYECTO 1

1.2 OBJETIVOS 3

1.3 METODOLOGÍA 4

CAPITULO 2

MARCO TEORICO 6

2.1 CONCEPTOS GENERALES DE SEGURIDAD 6

2.1.1 MODELO DE SEGURIDAD POSITIVO 6

2.1.2 MODELO DE SEGURIDAD NEGATIVO 7

2.1.3 FIREWALLS DE RED 7

2.1.4 FIREWALLS DE APLICACIÓN 9

2.2 PROTOCOLO HTTP 11

2.3 ATAQUES WEB CONOCIDOS 15

2.3.1 INJECTION FLAWS 15

2.3.2 HTML INJECTION 16

2.3.3 CROSS-SITE SCRIPTING (XSS) 17

2.3.4 FILE UPLOADS 17
2.3.5 REMOTE FILE INCLUSION 18
2.3.6 LOCAL FILE INCLUSION 18
2.3.7 SESSION HIJACKING 19
2.3.8 PARAMETER TAMPERING 19
2.4 APACHE HTTPD 20
2.4.1 DESCRIPCIÓN 20
2.4.2 FUNCIONAMIENTO COMO REVERSE PROXY 22
2.4.3 ETAPAS DE PROCESAMIENTO DE UNA PETICIÓN HTTP 23
2.5 MOD_SECURITY 25
2.5.1 DESCRIPCIÓN DEL MÓDULO 25
2.5.2 LENGUAJE DE REGLAS 29
2.6 SITUACIÓN ACTUAL DEL MERCADO 32

CAPITULO 3
IMPLEMENTACIÓN DEL PROYECTO 38
3.1 NÚCLEO DE DETECCIÓN DE ATAQUES 38
3.2 OPTIMIZACIÓN DE TRÁFICO WEB 48
3.2.1 CACHING DE PÁGINAS WEB 48
3.2.2 BALANCEO DE CARGA 51
3.2.3 OFFLOADING SSL 53
3.3 MANEJO DE SESIONES 56
3.3.1 SOPORTE DE REGISTRO DE SESIONES 56
3.3.2 CONTROL DE PUNTOS DE ENTRADA DE UN SITIO WEB 58
3.3.3 CREACIÓN DE UN ÁRBOL DE FLUJO DE UN SITIO WEB 59
3.4 ENMASCARAMIENTO DE COOKIES 62
3.5 AUTOAPRENDIZAJE 66
3.5.1 ANÁLISIS DE TRÁFICO 66

3.5.2 INTEGRACIÓN CON SCANNERS DE VULNERABILIDADES 72
3.6 ALTA DISPONIBILIDAD 77

CAPITULO 4
INTERFAZ DE MONITOREO 83
4.1 PROPÓSITO Y ARQUITECTURA DE LA INTERFAZ 83
4.2 FUNCIONALIDADES DE LA INTERFAZ 86

CAPITULO 5
PRUEBAS DE RENDIMIENTO 92
5.1 ESCENARIO DE PRUEBAS 92
5.2 ANCHO DE BANDA Y SESIONES CONCURRENTES 94

CONCLUSIONES
RECOMENDACIONES
ANEXOS
A. 1 REFERENCIA DE MODSECURITY
BIBLIOGRAFÍA

ABREVIATURAS

WAF	Web Application Firewall
SQLi	SQL Injection
XSS	Cross-site Scripting
RFI	Remote File Inclusion
LFI	Local File Inclusion
SQL	Standard Query Language
OS	Sistema Operativo
HTML	HyperText Markup Language
DLP	Data Loss Prevention
OWASP	The Open Web Application Security Project
CRS	Core Rule Set
SSL	Secure Sockets Layer
IDS/IPS	Intrusion Detection System/Intrusion Prevention System
TCP/IP	Transmission Control Protocol / Internet Protocol
NAT	Network Address Translation
FTP	File Transfer Protocol
MIME	Multipurpose Internet Mail Extensions
URL	Unified Resource Locator
HTTP	HyperText Transfer Protocol
URI	Unified Resource Identifier
LDAP	Lightweight Directory Access Protocol
PHP	PHP: Hypertext Preprocessor
IIS	Internet Information Services
ASM	Application Security Manager
PCI-DSS	Payment Card Industry – Data Security Standard
WAN	Wide Area Network
OSI	Open System Interconnection
XML	Extensive Markup Language
HTTPS	HTTP Secure
regex	Regular Expression
SDBM	Simple DataBase Manager
vpatch	Virtual Patch
WASC	Web Application Security Consortium
DNS	Domain Name System
CPU	Central Processing Unit

RAM	Random Access Memory
Mbps	Megabits por segundo
KBytes	Kilobytes

INDICE DE GRÁFICOS

Figura 2.1 Transacción HTTP típica 12
Figura 2.2 Ciclo de procesamiento de peticiones HTTP 24
Figura 3.1 Archivos de configuración del WAF 40
Figura 3.2 Ejemplo de servidores sin WAF 41
Figura 3.3 Ejemplo de servidores con WAF 42
Figura 3.4 Conversión HTTP a HTTPS 55
Figura 3.5 Inspección de tráfico HTTPS 55
Figura 3.6 Árbol de flujo natural de un sitio web 57
Figura 3.7 Enmascaramiento de cookies 65
Figura 3.8 Ejemplo de estructura de un sitio 68
Figura 3.9 Funcionamiento normal del cluster 80
Figura 3.10 Funcionamiento en failover 81
Figura 4.1 Arquitectura de interfaz de monitoreo 85
Figura 4.2 Buscador de alertas 87
Figura 4.3 Mapa de densidades de eventos 88
Figura 4.4 Eventos agrupados por hora 89
Figura 4.5 Clasificación de eventos por categorías 89
Figura 4.6 Distribución de ataques por país de origen 90
Figura 4.7 Diagrama de ataques 90
Figura 4.8 Inspección a fondo de eventos 91
Figura 5.1 Escenario de pruebas 93
Figura 5.2 Ancho de banda alcanzado sin keepalives 95
Figura 5.3 Sesiones concurrentes sin keepalives 95
Figura 5.4 Uso de CPU y RAM sin keepalives 96
Figura 5.5 Latencia por transacción HTTP sin keepalives 97
Figura 5.6 Ancho de Banda alcanzado con keepalives 98
Figura 5.7 Sesiones concurrentes con keepalives 98
Figura 5.8 Uso de CPU y RAM con keepalives 98
Figura 5.9 Latencia por transacción HTTP con keepalives 99

CAPITULO 1

ANTECEDENTES Y JUSTIFICACIÓN

1.1 DESCRIPCIÓN DEL PROYECTO

El presente proyecto tiene por finalidad el desarrollar un producto capaz de analizar todas y cada una de las partes de una transacción http y detectar posibles vectores de ataque a nivel de aplicaciones web. La importancia de esto radica en el hecho de que 70% de los ataques cibernéticos realizados hoy en día se enfocan justamente en las aplicaciones web alojadas en los servidores. Esto se debe a que no existe un ciclo de desarrollo de aplicaciones seguro, pues muchas veces los programadores no conocen de programación segura, o generalmente hay plazos de tiempo que cumplir que hacen que el

desarrollador se enfoque más en terminar todo aspecto que tenga que ver con funcionalidad y pase por alto la seguridad del sitio.

Las consecuencias de una mala programación de una aplicación web son grandes hoy en día debido a la criticidad de la información manejada por algunas de estas, pues la gente ingresa sus datos personales en algunas de ellas e incluso llega a ser hasta transacciones bancarias en línea, de modo que si alguna de estas aplicaciones permite a un atacante ver información de otros usuarios las consecuencias serían realmente graves.

Es aquí donde se vuelve imperativo tener un dispositivo que proporcione una capa adicional de seguridad a nuestras aplicaciones web de modo que la responsabilidad en cuanto seguridad del sitio no recaiga solamente en el programador, sino que se tenga además un filtro inteligente que permita detectar de manera proactiva ataques a nuestros sitios.

Un WAF nos brindaría protección en la capa de aplicación de manera análoga a la que un firewall brinda a nivel de capa de red. Tendremos un control mucho más granular que nos permitirá establecer reglas no

sólo en base a direccionamiento IP, sino elevarnos hasta la capa de aplicación y referirnos directamente a parámetros como información del payload POST de una petición HTTP, o cabeceras HTTP como el Referer o User-Agent enviados por el cliente, o incluso implementar restricciones en el contenido que un servidor pueda devolver a un cliente como Data Loss Prevention.

1.2 OBJETIVOS

- Desarrollar una plataforma completa para la protección de aplicaciones web por medio de la integración de herramientas Open Source disponibles y otras desarrolladas específicamente para el proyecto.
- Lograr una solución efectiva a nivel de costos para la protección de aplicaciones web y que ofrezca algunas de las funcionalidades ofrecidas por otras soluciones comerciales disponibles.
- Crear una interfaz de administración que facilite el uso del dispositivo de modo que el usuario no requiera de conocimientos extremadamente avanzados para el uso del dispositivo.
- Implementar una plataforma capaz de proteger varios sitios con niveles medios de tráfico sin afectar el nivel de servicio ofrecido por los servidores.

1.3 METODOLOGÍA

La plataforma estará basada en un servidor Apache httpd funcionando en modo de Proxy Reverso, lo que le permitirá ver todo el tráfico destinado a un sitio web. El núcleo del proyecto es un módulo de Apache llamado mod_security que permite el análisis de las peticiones y respuestas HTTP enviadas hacia y desde los servidores protegidos, así como el establecimiento de reglas para el filtrado del tráfico en base a parámetros a nivel de capa de aplicación. Se brindará protección contra ataques comunes "de paquete" utilizando un juego de reglas elaborado por OWASP llamado CRS (Core Rule Set) que incluye gran cantidad de firmas genéricas y específicas contra ataques web.

Adicional a esto se utilizarán e implementarán módulos adicionales que permitan brindar capacidades extra al dispositivo como balanceo de carga, offloading de SSL, caching de páginas web, transformación de HTTP a HTTPS, manejo de sesiones, enmascaramiento de cookies, enmascaramiento de software de servidores, prevención de fugas de información, auto-aprendizaje de comportamientos de sitios web y mecanismos de alta disponibilidad. Cada una de estas funcionalidades es detallada en capítulos posteriores.

Para la administración del equipo se proveerá una interfaz básica que permita a los usuarios abstraerse de la complejidad que implica un dispositivo de seguridad y presente opciones de configuración sencillas y enfocadas a la estructura de los sitios web protegidos, de modo que las configuraciones se basen en componentes conocidos por el administrador del sitio web a protegerse.

Para las pruebas de rendimiento se utilizarán herramientas de benchmarking de servidores web conocidas como httperf u otros generadores de tráfico HTTP que permitan evaluar la efectividad del WAF en base a parámetros como el ancho de banda, sesiones concurrentes, efectividad de autoaprendizaje y utilización de recursos del sistema. Adicional a esto se preparará un sitio web vulnerable de prueba al cual se pasará un scanner de vulnerabilidades web para verificar la eficacia del WAF al detener ataques.

CAPITULO 2

MARCO TEORICO

2.1 CONCEPTOS GENERALES DE SEGURIDAD

2.1.1 MODELO DE SEGURIDAD POSITIVO

Consistente en la idea de: "todo aquello que no está explícitamente permitido, está prohibido". Un ejemplo muy claro de este modelo es un cortafuegos de red "bien configurado" (aquél en el que permitimos el tráfico necesario mediante reglas explícitas y denegamos todo lo demás con la regla de limpieza al final de la configuración). Los mecanismos de autenticación (permiten el acceso a aquellos usuarios cuya contraseña/token/certificado es válido y deniegan a todos los demás); algunos WAF o Cortafuegos de Aplicaciones Web (definiendo una plantilla para una aplicación web en la que los parámetros han de tener un riguroso formato, de manera que si el usuario introduce

valores no permitidos, se bloquea el acceso) son ejemplos claros de este esquema.

2.1.2 MODELO DE SEGURIDAD NEGATIVO

Se basa en la idea de que todos los accesos a los recursos decididos están permitidos, excepto aquellos que sean prohibidos de forma explícita. Es decir: "todo lo que no está prohibido, está permitido". En esta filosofía podemos encuadrar a los antivirus (permito pasar todos los ficheros adjuntos a un correo electrónico excepto aquellos que contienen virus); los mecanismos antispam (todos los correos están permitidos excepto aquellos que lleven determinado contenido molesto); IDS/IPS (Intrusion Detection/Prevention Systems), algunos WAF(Web Application Firewalls, mediante la utilización de listas negras o de scoring, evitando los ataques conocidos a aplicaciones basadas en web: SQL Injections, Cross-Site Scripting, etc...).

2.1.3 FIREWALLS DE RED

Los firewalls o cortafuegos de red son dispositivos que se encargan de analizar el tráfico de red, y en base a reglas configuradas por el

administrador, permitir o denegar el paso del mismo. En particular el que se especifique que el firewall es de “red” se refiere al hecho de que su alcance de análisis involucra solamente aquellos datos que tienen que ver con la red, o dicho de otro modo aquella información obtenible analizando los encabezados de los protocolos hasta el nivel de transporte según el modelo OSI[1]. Si nos limitamos al uso de TCP/IP, un firewall de red sería capaz de tomar decisiones en base a direcciones IP y puertos de aplicación solamente.

Su función es principalmente la de controlar que IP puede comunicarse con que IP y determinar que puertos (servicios) le serán utilizables. Un firewall de red no puede, sin embargo, decidir si un paquete pasa o no en base a datos de la capa de aplicación como podría ser el URL solicitado en un paquete HTTP.

En cuanto al manejo de sesiones, un firewall de red puede llegar a mantener información hasta el nivel de sesiones TCP, permitiéndole ser un poco más granular en la definición de sus reglas. Por ejemplo se podría permitir todo el tráfico de salida de un servidor y de entrada

---

[1] Modelo utilizado como referencia para la explicación de los pasos efectuados en una comunicación de red.

solo permitir aquel que pertenezca a sesiones TCP iniciadas, denegando cualquier conexión entrante de manera efectiva.

Si por alguna razón se quisiera bloquear a un usuario (por ejemplo por generar gran cantidad de tráfico al servidor), a lo máximo que se podría llegar sería a bloquear su IP. Esto trae consecuencias debido al uso de mecanismos como NAT (Network Address Translation) que permiten que varios usuarios salgan enmascarados al internet con una IP en común. Suponiendo que se bloquea una IP de un usuario que está saturando el servidor, existiría la posibilidad de estar bloqueando a 100 usuarios mas que salen enmascarados con la misma IP al Internet.

2.1.4 FIREWALLS DE APLICACIÓN

Los firewalls de aplicación funcionan de manera similar a los firewalls de red pero con la única diferencia de que tienen un conocimiento a fondo de algún protocolo de nivel de aplicación (HTTP, FTP, MySQL, etc.) lo que les permite una granularidad mucho mayor al momento de crear reglas para el paso del tráfico. Por ejemplo, un firewall de aplicaciones web conoce a fondo el protocolo HTTP y permite

establecer reglas en base a parámetros de capa de aplicación también, por lo que se podría bloquear a cualquier cliente que ingrese la palabra SPAM en cualquier formulario en nuestra aplicación web, o no permitir el acceso a algún archivo en particular. Nótese que también es posible crear reglas que involucren parámetros de capas inferiores como la de red haciendo posible crear reglas en base a combinaciones de direcciones IP y parámetros HTTP.

Su función principal es la de proteger servidores que corren la aplicación para la que fueron diseñados. De este modo, un firewall de aplicaciones web tendrá por funcionalidad principal el proteger a un servidor web contra ataques provenientes de la red.

En el caso particular de un firewall de aplicaciones web se puede establecer reglas en base a sesiones por medio de los cookies enviados por el servidor. Esto nos permite crear reglas enfocadas específicamente a un usuario en particular por medio de su cookie de sesión, así como rastrear su actividad en el sitio. Esto da un control más fino que el manejo de sesiones TCP ofrecido por un firewall de red. Suponiendo que se quisiera bloquear a todo usuario que hiciera más de 100 peticiones al servidor en un segundo, sería posible

hacerlo por medio de cookies, bloqueando efectivamente al usuario atacante y a nadie más como ocurría con el NAT y el firewall de red.

2.2 PROTOCOLO HTTP

Desde 1990, el protocolo HTTP (Protocolo de transferencia de hipertexto) es el protocolo más utilizado en Internet. La versión 0.9 sólo tenía la finalidad de transferir los datos a través de Internet (en particular páginas Web escritas en HTML). La versión 1.0 del protocolo (la más utilizada) permite la transferencia de mensajes con encabezados que describen el contenido de los mensajes mediante la codificación MIME.

El propósito del protocolo HTTP es permitir la transferencia de archivos (principalmente, en formato HTML) entre un navegador (el cliente) y un servidor web. Cada transacción está identificada por un URL (Uniform Resource Locator) que indica al servidor el recurso solicitado por el cliente.

Desde el punto de vista de HTTP, la comunicación entre el navegador y el servidor se lleva a cabo en dos etapas:

- El navegador realiza una **solicitud o petición HTTP**

- El servidor procesa la solicitud y después envía una **respuesta HTTP**

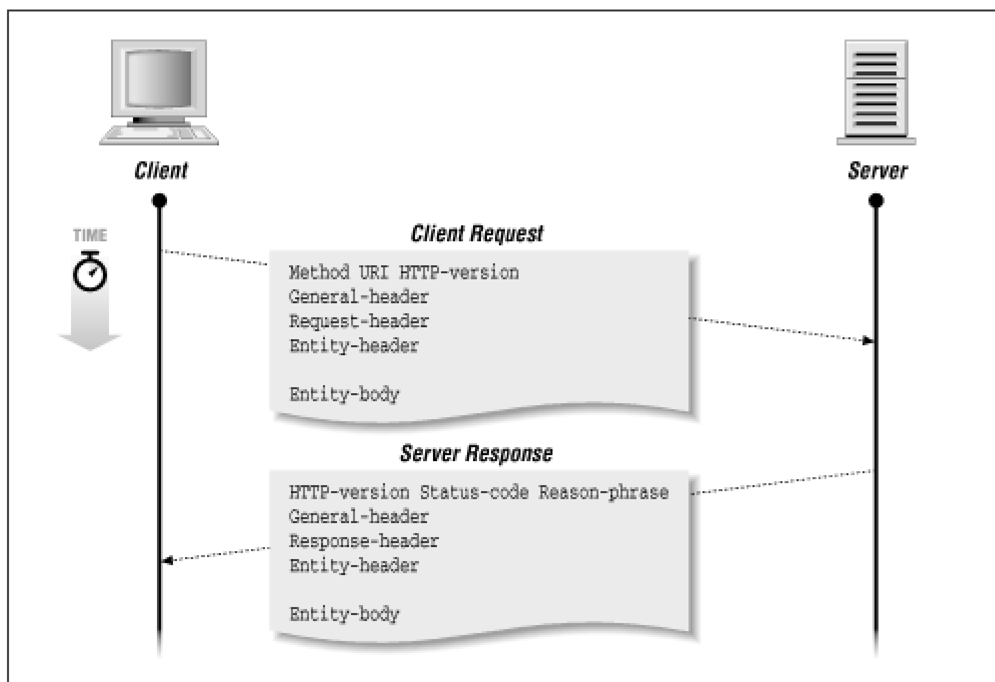

Figura 2.1 Transacción HTTP típica

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las **cookies**, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Una petición HTTP está formada básicamente por 5 partes:

- Método: Indica la manera en que el servidor deberá procesar la información enviada por el cliente. Los métodos más utilizados son GET y POST que indican básicamente la posición en la petición HTTP en la que el usuario podrá pasar datos al servidor (por ejemplo cuando envía un formulario al sitio web). GET envía la información de cliente como parte del URL, mientras que POST la envía en el cuerpo de la petición HTTP.
- URL: Cadena de caracteres que indica al servidor a que recurso se quiere acceder.
- Versión HTTP: Indica la versión del protocolo a utilizarse en la comunicación.
- Encabezados HTTP: Varios datos de cómo manejar la comunicación o los datos enviados.
- Cuerpo de la petición: Utilizado por el método POST para el envío de información desde el cliente hacia el servidor.

Una petición real se vería algo así:

```
GET /index.php HTTP/1.0
Host: www.example.com
User-Agent: nombre-cliente
```

Una respuesta típica de un servidor constaría principalmente de 4 partes:

- Versión HTTP: Indica la versión del protocolo a utilizarse en la comunicación.
- Código de respuesta: Código numérico de tres cifras que indica el estado de la petición hecha por el cliente que va seguido de un mensaje que describe el código. En general, todo código que comience por un 2 indicará una respuesta exitosa; todo código que comience por 3 indicará que el recurso buscado existe pero ha sido movido; Los códigos que comiencen por 4 o 5 representan errores de parte del cliente y del servidor respectivamente.
- Encabezados HTTP: Varios datos de cómo manejar la comunicación o los datos enviados.
- Cuerpo de la petición: Utilizado por el método POST para el envío de información desde el cliente hacia el servidor.

```
HTTP/1.1 200 OK
Date: Fri, 21 Dec 2012 00:00:00 GMT
Content-Type: text/html
Content-Length: 61

<html>
<body>
<h1>Hola mundo!!!</h1>
</body>
</html>
```

2.3 ATAQUES WEB CONOCIDOS

A continuación se presenta una breve explicación de los problemas más comunes de las aplicaciones web, pues se entiende que debemos conocer la estructura de un ataque común hacia un servidor web para saber cómo defenderse del mismo. El propósito de este capítulo no es tratar a profundidad en todas las variantes existentes de uno u otro ataque, sino de explicar la fundamentación teórica detrás de cada uno de ellos para poder contrarrestarlo. La lista de vulnerabilidades tratadas ha sido generada en base al proyecto OWASP Top Ten[2], que es un estudio acerca de los riesgos de seguridad en aplicaciones web de mayor impacto en la actualidad.

2.3.1 INJECTION FLAWS

Los fallos de inyección, tales como inyecciones SQL, OS, y LDAP ocurren cuando se permite el paso directo de datos ingresados por el usuario sin validar, desde la aplicación web hacia un intérprete cualquiera en el servidor, como parte de un comando o query. Los datos maliciosos enviados por el atacante engañan al intérprete haciendo que ejecute comandos indebidos o acceda a datos sin autorización.

[2] Se tomó como referencia la versión 2010

La manera de detectar dichos ataques es por medio del análisis minucioso de los parámetros enviados a la aplicación, en busca de patrones similares a la sintaxis común de los intérpretes vulnerables, como pueden ser bases de datos, o llamadas directas al Shell del sistema operativo.

Un esquema de seguridad positivo[3] ayuda a la prevención de estos ataques, aunque muchas veces sólo de manera parcial, gracias a la limitación en el tipo de caracteres que pueden enviarse en un parámetro, ya que algunos ataques requieren ciertos caracteres especiales para ser ejecutados, o gracias a la limitación en el tamaño de los parámetros, que limita la longitud de las cadenas de ataque válidas.

2.3.2 HTML INJECTION

Se da cuando una aplicación web utiliza total o parcialmente la información ingresada por un usuario para la generación dinámica de páginas. Si no se valida lo que el usuario ingresa, este podría ingresar código HTML en los parámetros, que sería utilizado en la generación de otras páginas que un usuario común vería como parte auténtica del

---

[3] Véase el capítulo 1.3

sitio web. Esto permitiría a un hacker abusar de la confianza que el usuario legítimo tiene en el sitio visitado, pidiéndole credenciales o haciendo anuncios falsos en nombre del sitio.

2.3.3 CROSS-SITE SCRIPTING (XSS)

Se trata de una variante de HTML injection en que se añade como parte del código reflejado al usuario algún script que se ejecute del lado del cliente, como puede ser javascript.

2.3.4 FILE UPLOADS

Se refiere al hecho de que un usuario pueda subir cualquier contenido al directorio público del sitio web de modo que éste sea visible a todo el mundo. Esto permitiría por ejemplo subir páginas dinámicas que interactúen directamente con el sistema (webshells), permitiendo a un atacante tomar control total del servidor. Este problema se da por no existir una verificación de los archivos subidos al servidor previo a su publicación.

2.3.5 REMOTE FILE INCLUSION

Esta vulnerabilidad es causada cuando un recurso en el servidor incluye a otro, pero la referencia del recurso incluido es manipulable por el cliente. En otras palabras, una página permite a un usuario incluir y ejecutar cualquier script en el Internet directamente desde el servidor vulnerable. Un atacante podría por ejemplo montar un servidor web X con un script en PHP que ejecute comandos del sistema, y explotar la vulnerabilidad en un servidor Y para correr ese script como si estuviera alojado en el servidor Y.

2.3.6 LOCAL FILE INCLUSION

Similar a un Remote File Inclusion, pues se da porque un script incluye a otro, pero ahora el script incluido se encuentra alojado en el servidor vulnerable. Un atacante podría por ejemplo incluir archivos del sistema de manera que estos se muestren en su browser (como el /etc/passwd en linux).

2.3.7 SESSION HIJACKING

Se da cuando un atacante es capaz de robar o adivinar una cookie que identifica una sesión válida y utilizarla para hacerse pasar por el usuario real. Esto permite al atacante ingresar a partes de un sitio que requieren autenticación sin necesidad de poseer credenciales válidas para la aplicación web.

2.3.8 PARAMETER TAMPERING

Se da cuando un sitio expone de manera incorrecta un parámetro interno de la aplicación web de modo que sea manipulable por el cliente, el cual puede alterar al funcionamiento normal de la página al falsificar el valor real del parámetro. Si una página de ventas en línea expusiera por alguna razón el precio de los ítems a comprar y utilizara de vuelta esa información para generar la factura, un cliente sería capaz de cambiar los precios de sus compras.

2.4 APACHE HTTPD

2.4.1 DESCRIPCIÓN

Apache HTTPD es el estándar de facto en software para montar servidores web en Linux (también existe una versión para Windows). La gran cantidad de módulos existentes permiten a un administrador de red personalizar el servidor a su medida, siendo una de las alternativas más flexibles en cuanto a funcionalidades se refiere. Esto en combinación con que es una alternativa completamente Open Source lo hacen el servidor web más utilizado en el mundo.

La arquitectura modular del servidor brinda combinaciones prácticamente infinitas que permiten dar funcionalidades diversas al servidor, como integrarlo con lenguajes de scripting como Perl, PHP, etc. De manera sencilla, o hacer que el servidor funcione como un Proxy web en cuestión de minutos. Permite esquemas de configuración para virtual hosting, de modo que un mismo servidor puede alojar múltiples sitios a la vez.

Para el alcance del documento, basta con saber la función de los siguientes módulos:

- mod_ssl: Provee al servidor apache de capacidades para el manejo de conexiones SSL, haciendo posible montar servidores que trabajen con HTTPS, dando cifrado en las comunicaciones entre cliente y servidor.
- mod_proxy: Permite al servidor funcionar como un Proxy HTTP, ya sea para su implementación como un Forward Proxy (Permite a usuarios ver cualquier página a través de él) o Reverse Proxy (Permite a los usuarios ver solamente ciertos sitios específicos).
- mod_headers: Permite modificar el contenido de las cabeceras HTTP, tanto de las peticiones como de las respuestas del servidor.
- mod_cache: Permite realizar caching de los sitios servidos, tanto en memoria como en disco para un servicio más rápido al encontrarse en modo de Proxy.

2.4.2 FUNCIONAMIENTO COMO REVERSE PROXY

Un reverse proxy se diferencia de un proxy común por el hecho de que en lugar de hacer frente a un grupo de clientes para controlar o verificar las peticiones de los mismos, un reverse proxy se sitúa del lado de los servidores para controlar el tráfico que proviene del mundo hacia los servidores. Es decir, su finalidad es frentear a los servidores y no a los clientes.

Las razones para establecer un proxy reverso podrían ser varias, como por ejemplo:

- Seguridad: el servidor proxy es una capa adicional de defensa y por lo tanto protege los servidores web.
- Cifrado / Aceleración SSL: cuando se crea un sitio web seguro, habitualmente el cifrado SSL no lo hace el mismo servidor web, sino que es realizado por el reverse proxy, el cual está equipado con un hardware de aceleración SSL (Security Sockets Layer).
- Distribución de Carga: el reverse proxy puede distribuir la carga entre varios servidores web. En ese caso, el reverse proxy

puede necesitar reescribir las URL de cada página web (traducción de la URL externa a la URL interna correspondiente, según en qué servidor se encuentre la información solicitada).

- Caché de contenido estático: Un reverse proxy puede quitar carga a los servidores web almacenando contenido estático como imágenes u otro contenido gráfico.

2.4.3 ETAPAS DE PROCESAMIENTO DE UNA PETICIÓN HTTP

Para entender funcionamiento de mod_security es importante saber en que puntos del procesamiento de una petición HTTP interfiere. Básicamente, el servidor apache sigue un ciclo para el procesamiento de peticiones de clientes tal y como se ilustra en la Figura 2.2.

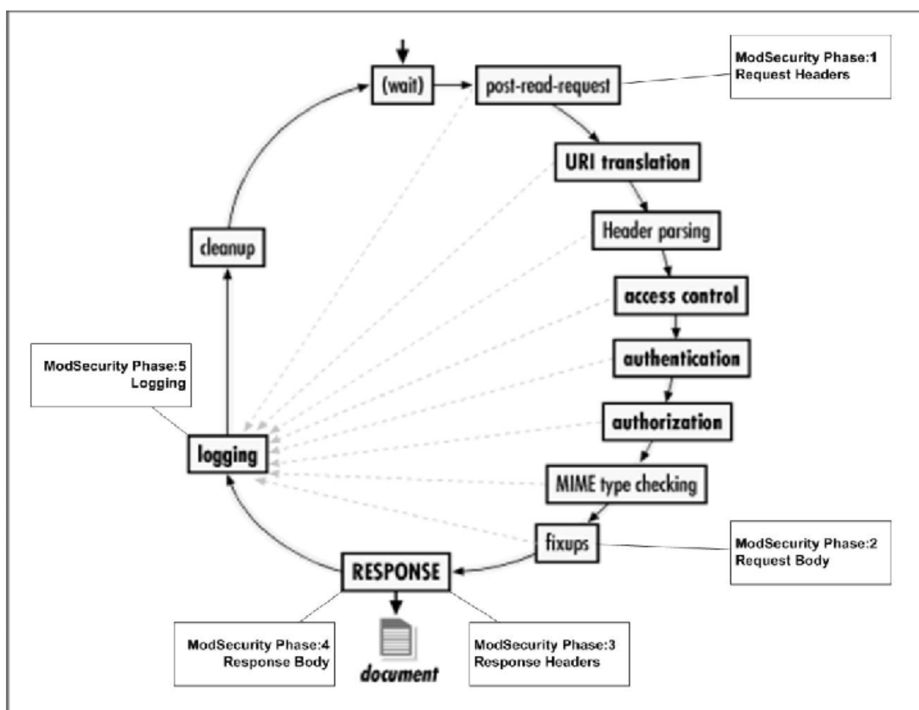

Figura 2.2 Ciclo de procesamiento de peticiones HTTP

Para lo que nos concierne, lo importante podría resumirse en los siguientes pasos:

1. Se recibe la petición HTTP del cliente.
2. Se procesan los encabezados HTTP de la petición.
3. Se verifican controles de acceso al recurso solicitado.
4. Se analiza el cuerpo de la petición HTTP que contiene los valores enviados por el usuario.
5. Se generan las cabeceras de la respuesta HTTP a ser enviada de vuelta al cliente.

6. Se procesa y genera el cuerpo de la respuesta que contiene la información solicitada por el cliente
7. Se generan logs de la acción realizada.

Hay que notar que dado que mod_security trabaja como un módulo de apache, solo dispondrá de la información que apache haya procesado. Es decir, no se tendrá información acerca del cuerpo de la petición mientras no se llegue al paso 4.

2.5 MOD_SECURITY

2.5.1 DESCRIPCIÓN DEL MÓDULO

Mod_security es un módulo de apache que permite la creación de reglas de filtrado de tráfico HTTP y que servirá como núcleo de este proyecto, pues será el componente activo en la detección de ataques contra los sitios web protegidos. Dicho de otro modo, mod_security incorpora las capacidades de un WAF en software en apache.

El proyecto de mod_security nació como una alternativa Open Source a los costosos appliances existentes en el mercado que proveen las capacidades de un WAF, buscando no solamente el abaratamiento de costos de implementación de este tipo de soluciones, sino proveer una

plataforma de similares capacidades a los dispositivos de otras marcas. Hoy en día, mod_security es una de las soluciones más utilizadas por su flexibilidad al momento de crear reglas y sus capacidades de detección.

Las ventajas de mod_security son varias, pues además de su lenguaje de reglas flexible permite:

- Obtención de logs detallados del tráfico HTTP: Normalmente los logs generados por los servidores web son útiles desde un punto de vista estadístico o enfocado a la parte comercial, mas no proveen información técnica suficiente como para una auditoría de logs en caso de un ataque al sitio web. No podemos por ejemplo verificar el contenido de las peticiones hechas por el usuario, de modo que si un ataque se llevo a cabo haciendo una petición con el método POST, no tendremos logs que respalden o comprueben que un ataque ocurrió. Mod_security nos permite mantener logs con absolutamente todas las partes de una transacción HTTP, de modo que podamos identificar vectores de ataque conocidos de una manera sencilla. Mod_security nos provee de un mecanismo en tiempo real contra amenazas de tipo web, cumpliendo funciones

similares a un IPS (Intrusion Prevention System) con la diferencia que se encuentra especializado en amenazas web, pues conoce el protocolo HTTP, lo que le permite abarcar mayor cantidad de amenazas.

Mod_security puede implementarse de dos maneras distintas: En la primera, el módulo protege el servidor apache en el que se instala. Todos los sitios alojados pueden ser protegidos de manera personalizada. Este modo se conoce como “modo embebido”. La segunda alternativa consiste en instalar mod_security sobre un reverse_proxy en apache, de modo que pueda proteger a todos los servidores web que se encuentren tras de él. La ventaja del segundo método está en que se vuelve posible proteger a otros tipos de servidores web como IIS o Tomcat, pues al funcionar como reverse proxy, el WAF solo interpreta el protocolo HTTP recibido de los servidores en el backend, independiente del software utilizado por ellos.

Mod_security se integra con apache por medio de 5 “ganchos” que se aferran a la ejecución del procesamiento de un request. Cada uno de

estos ganchos define lo que se conoce como una “fase” en el ciclo de procesamiento. Básicamente, las fases representan lo siguiente:

- Fase 1: En esta fase, mod_security cuenta con información correspondiente a las cabeceras de la petición HTTP hecha por el cliente. Una regla que trabaje en esta fase podrá involucrar solamente campos contenidos en las cabeceras de la petición.
- Fase 2: mod_security recibe de apache la información relacionada al cuerpo de la petición, de modo que aquí ya se pueden elaborar reglas que involucren datos enviados por el usuario hacia el servidor, como podrían ser los campos de un formulario.
- Fase 3: Esta fase se encuentra justamente después de la generación de los encabezados de la respuesta HTTP, por lo que hasta aquí podremos crear reglas que involucren encabezados de respuesta.
- Fase 4: En esta fase se ha terminado de generar el cuerpo de la respuesta. Normalmente aquí se colocan reglas que analicen el contenido devuelto por el servidor en busca de información sensible que no debiera ser pública normalmente. Estas reglas servirían por ejemplo para la implementación de Data Loss Prevention.

- Fase 5: Esta fase corresponde a la generación de logs por parte de apache y de mod_security. Normalmente no se crean reglas para esta fase.

2.5.2 LENGUAJE DE REGLAS

La configuración de reglas de mod_security se hace directamente a través del archivo de configuración de apache, en el que las reglas son escritas de manera secuencial. Las reglas serán procesadas de acuerdo a la fase en que se encuentren, de modo que todas las reglas de fase 1 se corren antes que las de fase 2, que a su vez se corren antes que las de fase 3 y así sucesivamente. El orden en que son escritas las reglas solo es relevante en el caso de las reglas que pertenecen a una misma fase, teniendo prioridad aquellas que son escritas primero.

La sintaxis de las reglas en mod_security es bastante sencilla:

SecRule *Target Operador* [*Acciones*]

El "Target" especifica que parte de la petición o respuesta HTTP se quiere chequear. Por ejemplo, podríamos especificar el nombre de la

variable “REQUEST_URI" que contiene el URL completo utilizado en una petición HTTP, sin incluir el nombre de dominio (por ejemplo “/index.php”).

El “Operador” permite establecer la relación buscada en la variable del “Target”. Existen un gran número de operadores utilizables, como los de igualdad numérica, mayor que, menor que, contención de una cadena en otra, hacer un phrase match con una lista de palabras, expresiones regulares, etc. El operador por defecto es el de expresiones regulares. Un ejemplo de operador sería “@rx spam” en el que se indica primeramente con una arroba el nombre del operador a utilizar (@rx equivale a usar expresiones regulares) seguido del valor a buscar. En el ejemplo se estaría buscando que en el contenido del target exista la palabra “spam”.

Finalmente, el parámetro de acciones es opcional y especifica la manera en la que se procederá si el target hace match con el operador. Las acciones a tomar indican si se desea generar logs por la alerta disparada, si la petición ha de bloquearse o no, si se desea insertar contenido adicional a la respuesta (como un mensaje de advertencia al usuario que disparo la alerta), dejar pasar el tráfico de

manera inmediata, etc. Existen acciones disruptivas y no disruptivas, que se diferencian porque si una acción disruptiva se ejecuta, se para el flujo de reglas de modo que no se siguen evaluando; una acción no disruptiva en cambio se ejecuta, pero no frena el flujo de reglas. Acciones disruptivas incluyen el bloqueo de un paquete o el paso inmediato del mismo, mientras que acciones no disruptivas podrían ser loguear un posible ataque detectado.

Existirán también circunstancias en que una sola regla no nos permite definir claramente el alcance del bloqueo que queremos realizar. El mayor problema con la sintaxis de las reglas es que no permite la comparación de varios targets con varios operadores en una misma regla, de modo que si quisiéramos especificar una regla que bloquee toda petición hecha a la URI "/index.php" y que contenga en uno de sus argumentos enviados por POST la palabra "ataque" no habría manera de plasmarlo en una regla. Es aquí donde entran los "chains", que nos permiten concatenar varias reglas de modo que acciones disruptivas se tomen solamente si todas las reglas de un "chain" hacen match. Para el ejemplo mencionado se tendría el siguiente chain:

```
SecRule REQUEST_URI "@rx ^/index.php$" "phase:2,log,chain"

SecRule ARGS "@rx ataque" "phase:2,log,auditlog,block"
```

Algo a notar en el ejemplo anterior es que el par de reglas mostrado puede evadirse fácilmente utilizando URL encoding. Si un usuario enviara como argumento “at%61que” en lugar de “ataque” la expresión regular no haría match. Para esto se incluyen en las acciones un grupo extenso de acciones no disruptivas que permiten realizar varias transformaciones al contenido del target antes de compararlo con el operador. Para decodificar el URL añadiríamos a la parte de acciones una transformación de tipo URL decode con la acción “t:urlDecode”. La aplicación de transformaciones es esencial en una buena regla, pues normalmente un atacante utilizará métodos de evasión para intentar pasar por alto los mecanismos de seguridad como un WAF.

2.6 SITUACIÓN ACTUAL DEL MERCADO

Al momento, mod_security es uno de los WAF más utilizados, normalmente basado en un modelo de seguridad negativo y mantenido por TrustWave. Si bien es una alternativa altamente flexible y de bajo costo, su mayor problema es que tiene una curva de aprendizaje bastante inclinada, lo que lo convierte en algo difícil de implementar de una manera completa. Las implementaciones actuales normalmente llegan al punto de utilizar el juego de firmas

proporcionado por OWASP que permite un grado de protección contra un gran número de ataques conocidos, pero se limita a un modelo de seguridad negativo. La implementación de un modelo de seguridad positivo es posible, pero requiere de alguien con conocimientos muy especializados del protocolo HTTP y del sitio web que se desea proteger, lo que rara vez es posible. El costo del motor de filtrado es nulo, pues se trata de una alternativa completamente Open Source. Otra desventaja es que al tratarse de un WAF en software, no implementa de forma nativa las características de aceleración de tráfico de un WAF en hardware.

En cuanto a soluciones comerciales existe una gran variedad de productos que se diferencian no solo en capacidades sino también en cuestión de precios. El rango es realmente abierto, pues existen soluciones basadas en Open Source que cuyos precios inician en $5000 hasta soluciones completamente propietarias que llegan a costar alrededor de los $60000.

Entre los proveedores de soluciones tenemos a Barracuda que provee dos líneas de productos distintas dependiendo de la escalabilidad requerida por el cliente: Por un lado está la línea de web site firewall enfocada a PyMEs con unidades que escalan entre los 25 y 100 Mbps

de throughput. La línea de web application controllers está enfocada a un mercado de mayores requerimientos. Los precios de los web site firewalls comienzan en $5500.

Por otro lado tenemos a Breach Security, marca recientemente adquirida por TrustWave quienes son los encargados de dar mantenimiento a ModSecurity, quienes proveen también un appliance completamente basado en la implementación en modo reverse proxy de la versión gratuita de mod_security para Apache. El nombre utilizado es WebDefend, y de acuerdo a Breach provee un motor de análisis bidireccional, perfilamiento en base a comportamientos del sitio, y múltiples motores de detección que trabajan colaborativamente para mantener seguros los sitios web protegidos. Los appliances trabajan tanto con modelos de seguridad negativos como positivos y su precio inicia en los $45.000.

F5 provee a su línea de equipos BigIP con un módulo para cumplir funciones de WAF. El nombre utilizado para referirse a dicha solución es “application security manager” (ASM). El ASM utiliza un modelo de seguridad positivo para la definición de sus políticas de seguridad y trae consigo toda posible característica deseada en un WAF

Enterprise. Cuenta además con integración opcional con el servicio de WhiteHat Sentinel Vulnerability Assessment Service que puede crear de manera automática parches en el ASM para vulnerabilidades detectadas en los sitios protegidos. El módulo de WAF tiene un costo estimado de $28000, mientras que la solución BigIP cuesta alrededor de $65000.

Finalmente tenemos a Imperva y su producto SecureSphere web application firewall que es considerado por Gartner como una de las mejores alternativas. El dispositivo usa una tecnología conocida como Transparent Inspection que le permite crear modelos de seguridad positivos. Imperva brinda opciones adicionales para la integración de otras soluciones como monitoreo de bases de datos o scanning de vulnerabilidades. Los precios comienzan alrededor de los $35000.

La acogida de los WAF no ha sido demasiado grande, pues existen gran cantidad de opositores a la idea y puristas que consideran que el aseguramiento de las aplicaciones web solo es efectivo completamente si se hace a nivel de código fuente, lo cual es completamente acertado pero poco aplicable a la realidad, pues si bien sería lo óptimo, hacer un cambio en el código fuente del sitio

puede llevar gran cantidad de tiempo, requerir de una ventana programada de trabajo, o simplemente no ser viable en caso de no conocer como se encuentra programada la aplicación. Un WAF permite realizar un parchado rápido y efectivo de una vulnerabilidad descubierta sin necesidad de entrar a la programación del sitio, lo cual sin duda alguna disminuye los tiempos de reacción inmediatos para remediar problemas de seguridad. Esto se conoce como Virtual Patching.

Una de las principales razones por las que los WAF se han dado a conocer es el cumplimiento de la norma PCI-DSS (Payment Card Industry Data Security Standard) que desde su versión 1.1 establece en su requerimiento 6.6 que toda organización necesita proteger sus aplicaciones web ya sea por medio de revisiones de código periódicas (auditorías que resultan costosas) o bien instalando un WAF. Si bien es cierto que lo ideal sería implementar ambos métodos, la norma solo requiere uno de ellos, por lo que muchas compañías han optado por implementar un WAF por cuestiones de conveniencia.

La esperanza sin embargo es que en un futuro próximo todo sitio web cuente con algún mecanismo de protección especializado, sea este

implementado en software o hardware, por lo que se espera que la necesidad de este tipo de soluciones crezca en gran manera y abarate costos, y dado el alto crecimiento de los ataques web de hoy en día es más que seguro que esto se cumpla.

CAPITULO 3

IMPLEMENTACIÓN DEL PROYECTO

3.1 NÚCLEO DE DETECCIÓN DE ATAQUES

El WAF no es más que un servidor Apache funcionando en modo Reverse Proxy, que intercepta el tráfico HTTP dirigido a los servidores que protege y lo analiza en busca de patrones maliciosos. Las decisiones tomadas por el WAF en cuanto a si el tráfico analizado es o no legítimo dependerá de las reglas configuradas en él a través de mod_security que es un módulo de Apache que implementa las funcionalidades básicas de un WAF. Mod_security provee al usuario de un lenguaje flexible para el establecimiento de reglas para el análisis del tráfico web basado en la capacidad de comparar parámetros de los paquetes http con patrones definidos por el usuario,

escritos por medio de expresiones regulares. Así podremos comparar, por ejemplo, los argumentos enviados en un request GET con un conjunto de palabras características de ataques comunes y desechar estos paquetes antes de que lleguen al servidor, brindando una protección proactiva y en tiempo real de nuestros servidores web.

ESTRUCTURA DE LOS ARCHIVOS DE CONFIGURACIÓN

El archivo de configuración principal se encuentra en la siguiente dirección:

```
/etc/httpd/conf/httpd.conf
```

Y contiene la configuración básica de cualquier servidor Apache y a su vez incluye todo archivo de extensión .conf contenido en la carpeta:

```
/etc/httpd/conf.d/
```

El árbol de inclusiones sería el siguiente:

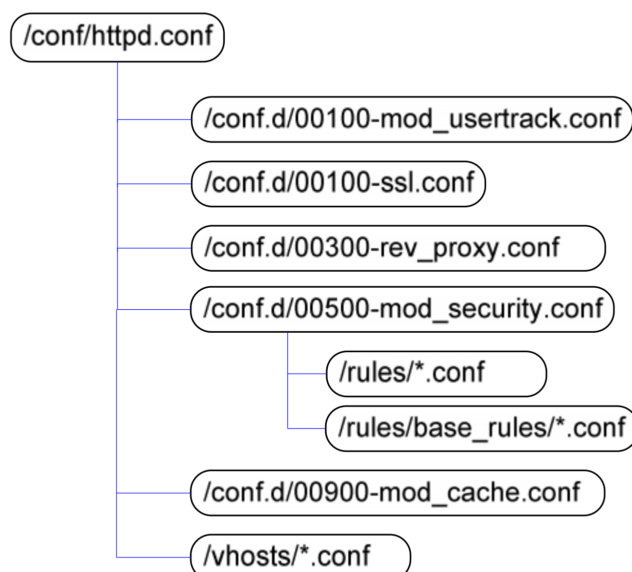

Figura 3.1 Archivos de configuración del WAF

INSTALACIÓN DE APACHE COMO REVERSE PROXY

Para que Apache funcione como Reverse Proxy, basta con activar el módulo mod_proxy que brinda las capacidades de funcionar tanto como Forward Proxy o Reverse Proxy. En este caso nos enfocaremos en el segundo modo. Dado que mod_proxy viene por defecto con Apache, lo único que debemos tener en cuenta es que el módulo se esté cargando. La configuración del Proxy se encuentra en el archivo:

```
/conf.d/00300-rev_proxy.conf
```

Buscamos en el archivo las líneas siguientes:

```
LoadModule proxy_module modules/mod_proxy.so

LoadModule proxy_http_module modules/mod_proxy_http.so
```

```
LoadModule proxy_connect_module modules/mod_proxy_connect.so
```

Donde la primera línea activa el mod_proxy y la segunda le da capacidades para ser un proxy HTTP. Por defecto, al activar estos módulos, tendremos funcionando un Forward Proxy u Open Proxy que cualquier persona autorizada o no podrá usar como túnel para hacer peticiones a cualquier sitio en Internet. Para deshabilitar esta funcionalidad y que sólo trabaje como Reverse Proxy escribiremos la siguiente línea:

```
ProxyRequests Off
```

Ahora tendremos que configurar los Hosts-Virtuales en el WAF, uno por cada sitio que se proteja, pero antes debemos tener en cuenta que como el WAF funcionará como Reverse Proxy deberá suplantar las IPs originales de los servidores, de tal manera que sirva como punto de presencia. Poniendo un ejemplo: en un comienzo nuestros servidores estarán conectados de la siguiente manera:

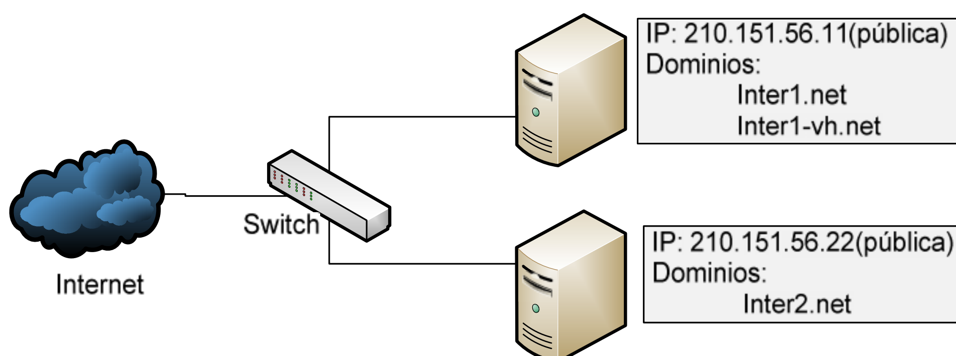

Figura 3.2 Ejemplo de servidores sin WAF

Luego de conectar el WAF tendremos la siguiente topología:

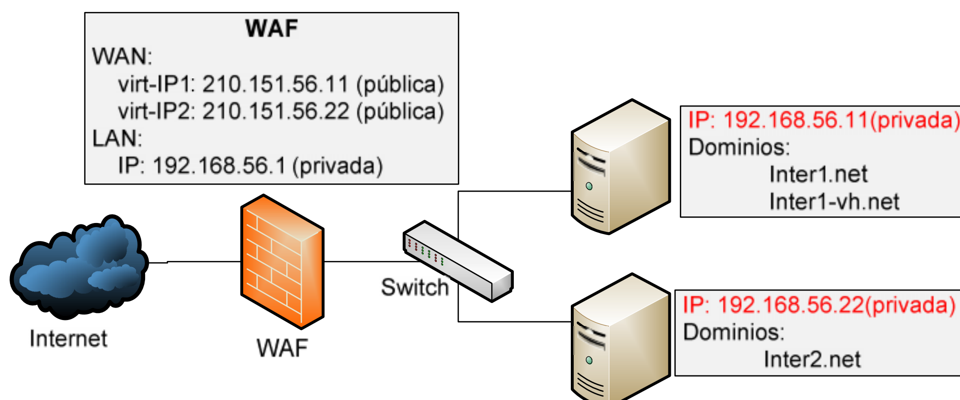

Figura 3.3 Ejemplo de servidores con WAF

Podemos ver que las direcciones públicas que antes pertenecían a los servidores son ahora de interfaces virtuales en el puerto WAN del WAF, mientras que los servidores ahora tienen IPs privadas. La creación de sub-interfaces en el WAF se hace añadiendo un archivo en

```
/etc/sysconfig/network-scripts/
```

Cuyo nombre sea ifcfg-ethX:Y donde X representa el número de la interfaz real, y Y el identificador de la sub-interfaz. El archivo deberá contener lo siguiente (reemplazando los valores necesarios):

```
TYPE=Ethernet
DEVICE=eth0:0
BOOTPROTO=none
NETMASK=255.255.255.0
IPADDR=210.151.56.11
```

Para el ejemplo anterior debería configurarse otra interfaz virtual con la dirección del otro servidor. Adicional a esto se debe recordar que el único propósito del WAF es filtrar el tráfico web y ningún otro, por lo que deberá usarse iptables para el mapeo de direcciones públicas a privadas para hacer un bypass de aquellos puertos no usados por el servidor web:

```
iptables -t nat -A PREROUTING -d 210.151.56.11/32 -p tcp --dport 80 -j ACCEPT

iptables -t nat -A PREROUTING -d 210.151.56.11/32 -p udp --dport 80 -j ACCEPT

iptables -t nat -A PREROUTING -d 210.151.56.11/32 -p tcp --dport 443 -j ACCEPT

iptables -t nat -A PREROUTING -d 210.151.56.11/32 -p udp --dport 443 -j ACCEPT

iptables -t nat -A PREROUTING -d 210.151.56.11/32  -j DNAT --to-destination 192.168.56.11
```

Una vez hecho esto podemos configurar los hosts virtuales, cada uno de los cuales tendrá un archivo separado de configuración (extensión .conf) en la carpeta

```
/vhosts/
```

Siguiendo el ejemplo de arriba, se configurarán los 3 dominios mostrados en el gráfico:

Inter1.net.conf:

```
<VirtualHost 210.151.56.11:80>

      ServerName inter1.net

      ProxyPreserveHost On

      ProxyPass / http://192.168.56.11/

      ProxyPassReverse / http://192.168.56.11/

</VirtualHost>
```

Inter1-vh.net.conf:

```
<VirtualHost 210.151.56.11:80>

      ServerName inter1-vh.net

      ProxyPreserveHost On

      ProxyPass / http://192.168.56.11/

      ProxyPassReverse / http://192.168.56.11/

</VirtualHost>
```

Inter2.net.conf:

```
<VirtualHost 210.151.56.22:80>

      ServerName inter2.net

      ProxyPreserveHost On

      ProxyPass / http://192.168.56.22/

      ProxyPassReverse / http://192.168.56.22/

</VirtualHost>
```

Los parámetros de configuración de cada host-virtual tienen las siguientes funciones:

- ServerName.- Identifica el nombre de dominio del Host-Virtual.
- ProxyPass y ProxyPassReverse.- Se encargan de hacer el mapeo de direcciones públicas a privadas y viceversa.
- ProxyPreserveHost.- Indica al proxy que no cambie la cabecera HTTP "Host:" recibida originalmente al pasarla a los servidores.

Dado que se están manejando varios Hosts-Virtuales en una misma IP (210.151.56.11), se tendrá que indicar al Apache que distinga los Hosts por el valor de la cabecera "Host:" enviado por el cliente. Esto se hace agregando la siguiente línea al archivo de configuración principal:

```
NameVirtualHost 210.151.56.11:80
```

INSTALACIÓN DE MOD_SECURITY

Para instalar mod_security basta con bajarse la última versión del código fuente y compilarlo e instalarlo. Antes de continuar con la instalación deberemos asegurarnos de tener instalados los siguientes paquetes:

- libapr
- libapr_util
- libxml2
- libpcre

Adicionalmente debemos activar el módulo de Apache llamado mod_unique_id que viene por defecto con Apache. Para continuar, nos ubicamos en la carpeta del mod_security, abrimos la carpeta apache2 y escribimos los siguientes comandos:

```
./configure

make

make install
```

Por último, cargamos el soporte xml y el mod_security agregando las siguientes líneas al archivo de configuración del Apache:

```
LoadFile /usr/lib/libxml2.so

LoadModule security2_module modules/mod_security2.so
```

Hasta aquí ya debemos tener el mod_security funcionando sobre el reverse proxy, pero sin ninguna regla configurada.

DETECCION DE ATAQUES COMUNES: CORE RULE SET

Actualmente, el mod_security incluye un juego de reglas que proveen protección contra una gran cantidad de ataques conocidos e irregularidades en las cabeceras HTTP, de modo que podamos establecer un esquema de seguridad negativo desde el momento en que instalemos mod_security. Para poner a trabajar el Core Rule Set (CRS) debemos dirigirnos a la carpeta del instalador de mod_security, donde habrá una carpeta llamada “rules”, la que debemos copiar a:

```
/etc/httpd/
```

Al abrirla encontraremos 2 archivos de configuración:

- modsecurity_crs_10_global_config.conf que contiene las directivas de configuración globales de mod_security, y que deberá ser incluido en el archivo de configuración:

```
/conf.d/00500-mod_security.conf
```

- modsecurity_crs_10_config.conf que contiene configuraciones específicas para incluirse en Hosts-Virtuales y que servirá como plantilla para establecer las configuraciones de cada uno de ellos. Este archivo deberá copiarse y editarse para luego ser incluido en cada contenedor de host-virtual con las configuraciones ajustadas a la medida de cada uno de ellos.

Adicionalmente existirán dos carpetas que contienen varios archivos, cada uno de los cuales representa un conjunto de reglas específico para prevenir ciertos tipos de ataques que deberán ser incluidos en cada contenedor de host-virtual dependiendo de los requerimientos de cada aplicación web.

3.2 OPTIMIZACIÓN DE TRÁFICO WEB

Dado que estamos trabajando con un reverse proxy, resultaría conveniente implementar funcionalidades extra que permitan ampliar el campo de aplicación del WAF como Caching de páginas web, balanceo de carga entre varios servidores y conversión transparente de HTTP a HTTPS.

3.2.1 CACHING DE PÁGINAS WEB

Apache trae por defecto un módulo llamado mod_cache que permite implementar las funcionalidades de un proxy cache ya sea en memoria o en disco. El mayor problema con la implementación de cache en memoria de Apache es que para cada proceso nuevo, el servidor

copiará completamente la porción de memoria del cache, ocupando de manera innecesaria memoria que será bastante requerida por mod_security. Por las anteriores razones se optó por el modo de caching en disco.

Todas las configuraciones del cache están en el archivo 00900-mod_cache.conf. Abrir el archivo y confirmar que el módulo se está cargando en el servidor, para lo cual deberán existir las siguientes líneas:

```
LoadModule cache_module modules/mod_cache.so

LoadModule disk_cache_module modules/mod_disk_cache.so
```

La primera línea carga mod_cache y la segunda le permite hacer caching en disco. La configuración de mod_cache está dada por los siguientes parámetros:

```
<IfModule mod_cache.c>
      CacheDefaultExpire 3600
      CacheMaxFileSize 128000
      CacheMinFileSize 64

      <IfModule mod_disk_cache.c>
            CacheRoot /etc/httpd/cache/
            CacheEnable disk /
            CacheDirLevels 3
            CacheDirLength 1
      </IfModule>
```

```
</IfModule>
```

- CacheDefaultExpire especifica el tiempo de expiración asignado por defecto (en caso de no especificarse en las cabeceras HTTP) en segundos.
- CacheMaxFileSize y CacheMinFileSize especifican el tamaño máximo y mínimo de los archivos guardados en cache en bytes.
- CacheRoot especifica la ruta en que se guardaran los archivos cacheados.
- CacheEnable habilita el caching de páginas. El primer parámetro el el tipo de caching (disk) y el segundo es la ruta de la página que será cacheable (/).
- CacheDirLevels especifica el número de niveles en la jerarquía de carpetas que se crearán dentro de la raíz del cache (especificado por CacheRoot).
- CacheDirLength especifica la longitud de los nombres de carpetas usados para crear la jerarquía de directorios de caching.

3.2.2 BALANCEO DE CARGA

Apache trae por defecto el módulo mod_proxy_balancer que permite definir clusters de servidores de manera sencilla. Cada cluster se crea en un archivo de configuración diferente dentro de la carpeta /vhosts/ que deberá contener lo siguiente:

```
<Proxy balancer://cluster1 stickysession=JSESSIONID|jsessionid>

      BalancerMember http://10.0.1.50 loadfactor=2

      BalancerMember http://10.0.1.51 loadfactor=5

      BalancerMember http://10.0.1.52 status=+H

</Proxy>
```

Donde cluster1 es un nombre cualquiera dado al cluster, que puede ir seguido de parámetros de configuración del cluster como stickysession que permite especificar una lista de nombres de cookies utilizados para identificar la sesión de usuario y dirigir los requests de un mismo usuario siempre al mismo servidor del cluster de modo que la sesión no se pierda, y la directiva BalancerMember permite añadir un miembro al cluster, recibiendo como parámetros la dirección del miembro del cluster seguida de parámetros adicionales de configuración. En este caso se utilizan loadfactor que permite establecer un peso para la distribución de requests, de modo que en el

ejemplo, el primer miembro recibe 2 requests por cada 5 requests recibidos por el segundo miembro; el parámetro status permite especificar información adicional del miembro, como en este caso que se configura al tercer miembro como hotseat, lo que significa que sólo será utilizado si ninguno de los otros miembros está disponible. Para una lista completa de parámetros consultar la documentación de mod_proxy en:

```
http://httpd.apache.org/docs/2.2/mod/mod_proxy.html
```

Para direccionar un host-virtual al cluster, se deberán reemplazar las direcciones privadas por balancer://cluster1 (o el nombre utilizado en la definición del cluster):

```
<VirtualHost 210.151.56.11:80>
      #Cambio en las dos siguientes líneas
      ProxyPass / balancer://cluster1/
      ProxyPassReverse / balancer://cluster1/
      ServerName inter.net
      ProxyPreserveHost On
</VirtualHost>
```

3.2.3 OFFLOADING SSL

Hoy en día existen aplicaciones web que manejan contenido crítico, como información personal, cuentas bancarias, seguridad social y demás información sensible que no debe ser conocida por nadie más que el cliente al que pertenece. El problema con el protocolo HTTP es que transporta todos los datos en texto plano, permitiendo a cualquiera que intercepte el paquete descubrir lo que lleva adentro. Por esta razón se procedió a utilizar conexiones cifradas con SSL que permitan ocultar la información de quien no deba verla.

Desde el punto de vista del WAF, el tráfico que pase encriptado por el canal será imposible de analizar, a no ser que este cuente con las llaves pública y privada de los servidores que protege. Lo interesante está en que teniendo las llaves, se podría ofrecer SSL offloading desde el WAF, que consiste en descifrar todo el tráfico dirigido hacia el servidor directamente sobre el WAF, liberando recursos del servidor web. Adicionalmente existe la nueva posibilidad de convertir tráfico HTTP puro a HTTPS por medio del WAF de una manera transparente tanto para el cliente como para el servidor.

En ambos casos, las funcionalidades pueden implementarse utilizando mod_ssl y activando el motor SSL en los virtual hosts que se quiera proteger. La configuración mínima requerida para un virtual host es la siguiente:

```
## Se activa SSL en el vhost
SSLEngine on
## Se configuran los algoritmos de cifrado y versiones de
## SSL a utilizar
SSLProtocol all -SSLv2
SSLCipherSuite    ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:-LOW:-
EXP:-NULL
## Se indica al vhost las llaves pública y privada para
## la infraestructura PKI
SSLCertificateFile /etc/ssl/certs/server.crt
SSLCertificateKeyFile /etc/ssl/certs/server.key
```

Se debe tener en cuenta que en caso de realizar una conversión de HTTP a HTTPS, las conexiones realizadas por el WAF consisten en HTTPS entre el cliente y el WAF, y HTTP puro entre el WAF y el servidor. El WAF deberá tener un certificado SSL válido con el nombre del servidor al que protege.

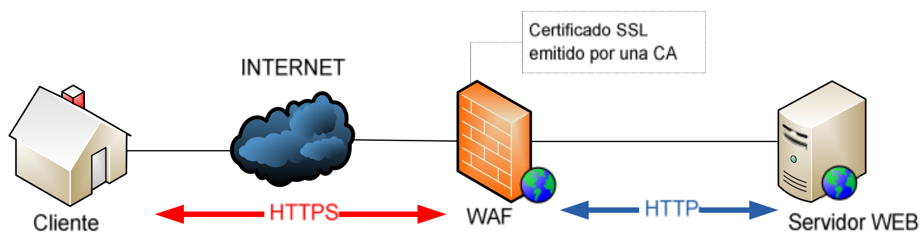

Figura 3.4 Conversión HTTP a HTTPS

En caso que se requiera HTTPS en todo el trayecto entre el cliente y el servidor, el certificado y llave privada original del servidor deberá pasar al WAF ya que este será quien reciba y descifre el tráfico HTTPS enviado por el cliente, luego lo vuelva a cifrar y lo envíe por medio de otra conexión HTTPS que puede tener un certificado interno autogenerado ya que no será visto por los clientes.

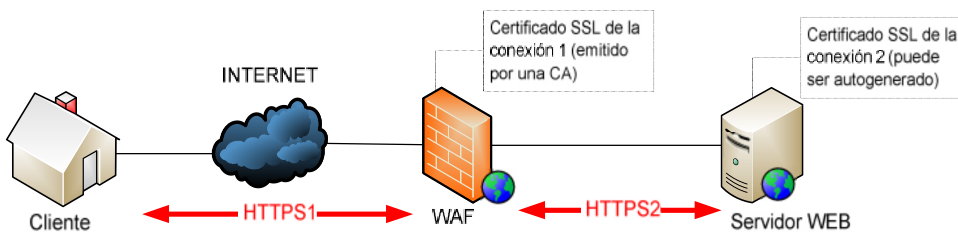

Figura 3.5 Inspección de tráfico HTTPS

3.3 MANEJO DE SESIONES

3.3.1 SOPORTE DE REGISTRO DE SESIONES

Mod_security provee la funcionalidad de manejar sesiones por medio de colecciones, que no son más que variables que persisten a través de varios peticiones y son identificados por medio de un cookie determinado. Para inicializar la colección SESSION utilizamos las siguientes reglas:

```
SecRule REQUEST_COOKIES:PHPSESSID !^$ chain,nolog,pass
SecAction setsid:%{REQUEST_COOKIES.PHPSESSID}
```

Esto nos permite utilizar a partir de ahora la colección para pasar variables entre peticiones correspondientes a la misma sesión, dándonos un control a nivel de usuario en cuanto a los bloqueos y establecimientos de reglas. Una de las capacidades que podría implementarse con esto son puntos de entrada reglamentarios a un sitio. Es decir, un usuario no podrá visitar determinados recursos de una aplicación web sin tener una sesión activa antes.

Bajo este criterio podríamos también implementar un árbol de accesos para los recursos que controlen el flujo de un usuario a través de un determinado sitio. Por ejemplo, restringir desde que recursos podemos acceder a otros, es decir, a la página 2 sólo podemos llegar si antes pasamos por la página 1; a las páginas 3 o 4 sólo si veníamos de la 2, limitando el flujo de un usuario a algo que parezca una navegación típica humana. Esto podría ayudar a detener ciertos intentos de crawling o ataques de fuerza bruta que intenten acceder de manera secuencial a varios recursos, sin seguir el flujo natural de links.

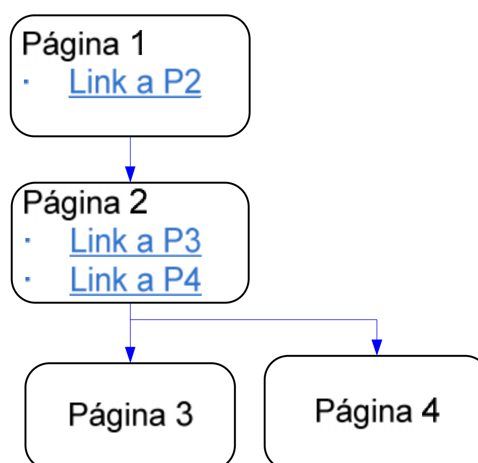

Figura 3.6 Árbol de flujo natural de un sitio web

3.3.2 CONTROL DE PUNTOS DE ENTRADA DE UN SITIO WEB

Se puede definir un conjunto de reglas que obliguen a los usuarios a visitar una página en particular (normalmente la página de entrada al sitio) antes de poder acceder al resto de recursos de la aplicación web. Esto nos permite establecer puntos de ingreso obligatorios al sitio para asegurarnos que cualquier persona deba tener una sesión activa para visitar todos los contenidos del sitio, permitiéndonos un control más preciso de lo que hace. Esto se puede hacer por medio de un conjunto de reglas sencillo con el siguiente esquema:

```
## iniciamos la colección SESSION
SecRule REQUEST_COOKIES:SECookie2 "!^$" "chain,nolog,pass"
SecAction "setsid:%{REQUEST_COOKIES.SECookie2}"

## Si la sesión es nueva, se setea la variable active
## de la colección session a 1.
SecRule SESSION:IS_NEW "@eq 1" "setvar:session.active=1"

## Si la sesión está activa, dejar
## pasar el request incondicionalmente
SecRule SESSION:ACTIVE "1" "phase:2, skipAfter:969696"

## En caso de hacerse un request a index.php,
```

```
## dejarlo pasar y setear
## la variable active de la sesión a 1.
SecRule REQUEST_FILENAME "^/(index.php)?$" \
"phase:2,       t:none,       t:lowercase,       t:normalisePath,
skipAfter:969696,     setvar:session.active=1"

## Si no existe sesión activa y el request no es a la página de
## ingreso del sitio, denegar incondicionalmente el request.
SecAction "phase:2,deny,status:403"
SecMarker 969696
```

3.3.3 CREACIÓN DE UN ÁRBOL DE FLUJO DE UN SITIO WEB

Si se requiere, puede limitarse los puntos de acceso a cada recurso de un sitio web dependiendo del recurso anteriormente visitado. Así se puede establecer un árbol de flujo que restrinja la movilidad de un visitante del sitio web a un modelo humano (normalmente un ser humano seguirá solamente links en el sitio) y prohibir a ciertos robots el crawling automatizado del sitio. A continuación un ejemplo de una estructura en la que se tiene una cadena de páginas enumeradas donde para entrar a la 2 se debe venir de la 1; para entrar a la 3 se

debe venir de la 2; y así sucesivamente. Nótese que para el ejemplo se utiliza el cookie “SECookie2” para diferenciar las sesiones.

```
## iniciamos la colección SESSION
SecRule REQUEST_COOKIES:SECookie2 "!^$" \
        "chain,nolog,pass"
SecAction "setsid:%{REQUEST_COOKIES.SECookie2}"

## Si la sesión es nueva, se asume que la última página visitada fue
## index.php
SecRule SESSION:IS_NEW "1" \
        "setvar:session.lpage=index.php"
```

```
## Si se ingresa al index.php, se acepta el request y se setea
## la última página visitada a index.php
SecRule REQUEST_FILENAME "^/(index\.php)?$"
        "phase:2,t:none,t:lowercase,t:normalisePath,skipAfter:969696,log,setvar:session.lpage=index.php"

## Si se solicita 1.php se verifica que la página anterior sea
## index.php o 1.php, caso contrario se deniega el acceso
SecRule REQUEST_FILENAME "!^/1\.php$" \
        "phase:2,t:none,t:lowercase,t:normalisePath,skip:1,nolog"
SecRule SESSION:LPAGE "^(index|1)\.php$" \
        "t:none,nolog,setvar:session.lpage=1.php,skipAfter:969696"


## Si se solicita 2.php se verifica que la página anterior sea
```

```
## 1.php o 2.php, caso contrario se deniega el acceso
SecRule REQUEST_FILENAME "!^/2\.php$" \
	"phase:2,t:none,t:lowercase,t:normalisePath,skip:1,nolog"
SecRule SESSION:LPAGE "^(1|2)\.php$" \
	"t:none,nolog,setvar:session.lpage=2.php,skipAfter:969696"

## Si se solicita 3.php se verifica que la página anterior sea
## 2.php o 3.php, caso contrario se deniega el acceso
SecRule REQUEST_FILENAME "!^/3\.php$" \
	"phase:2,t:none,t:lowercase,t:normalisePath,skip:1,nolog"
SecRule SESSION:LPAGE "^(2|3)\.php$" \
	"t:none,nolog,setvar:session.lpage=3.php,skipAfter:969696"

## Si se solicita 4.php se verifica que la página anterior sea
## 3.php o 4.php, caso contrario se deniega el acceso
SecRule REQUEST_FILENAME "!^/4\.php$" \
	"phase:2,t:none,t:lowercase,t:normalisePath,skip:1,nolog"
SecRule SESSION:LPAGE "^(3|4)\.php$" \
	"t:none,nolog,setvar:session.lpage=4.php,skipAfter:969696"

## Si se solicita cualquier otra página se deja pasar el
requerimiento
SecRule REQUEST_FILENAME "!^(/index|/1|/2|/3|/4)\.php$" \
  "phase:2,t:none,t:lowercase,t:normalisePath,skipAfter:969696,nolog"

## Finalmente se deniega todo el resto de casos
SecAction "phase:2,deny,status:403"

## Este marcador indica el fin del árbol de flujo del sitio web
```

SecMarker 969696

3.4 ENMASCARAMIENTO DE COOKIES

En ciertas aplicaciones se da un uso inapropiado a los cookies, haciendo que estos contengan datos sensibles o que su contenido influya en la creación de las páginas dinámicas de un sitio. En el primer caso, esto podría permitir el robo de datos si alguien lograra acceder al cookie, mientras que en el segundo caso podría permitir (dependiendo de la aplicación) que se efectúen ataques de tipo XSS u otros dependiendo del uso del cookie. El problema realmente está en que el usuario puede modificar el contenido del cookie antes de enviarlo al servidor, haciendo que la aplicación pueda funcionar de maneras inesperadas. Lo que se hace entonces es enmascarar el valor real de los cookies enviados por los servidores con datos aleatorios sin relación con el contenido original. Esto previene las fugas de información y protege al servidor de los casos en que un usuario manipula un cookie, ya que el WAF se encarga de descartar toda cookie que no haya emitido.

Para añadir esta funcionalidad se hizo un módulo de Apache que se encarga de reemplazar las cookies que envía el servidor por cadenas de longitud configurable, y hacer el mapeo inverso correspondiente. El mapeo de cookies internas y externas (entiéndase por internas las cookies originales enviadas por los servidores web, y por externas aquellas enmascaradas) se mantiene dentro de una base de datos SDBM.

El módulo tiene que encargarse de 2 procesos: enmascarar todo cookie contenido en una cabecera Set-Cookie enviada por el servidor en una respuesta, y desenmascarar todo cookie utilizado por el cliente en los encabezados Cookie de una petición.

El proceso llevado a cabo por el módulo para enmascarar un cookie es el siguiente:

1. El servidor envía una respuesta que contiene un encabezado Set-Cookie con uno o varios cookies.
2. El WAF analiza la respuesta y genera cadenas aleatorias para reemplazar los cookies en el encabezado Set-Cookie.

3. El WAF almacena en una base SDBM el cookie interno con un timestamp en un registro cuya llave es la cadena aleatoria generada en el paso 2.
4. El WAF envía la respuesta con cookies modificados al cliente.

Para desenmascarar un cookie el proceso es este:

1. El cliente envía un request que contiene un encabezado Cookie.
2. El WAF busca en la base de datos SDBM si existe el cookie.
3. Si el cookie no existe lo descarta y envía la petición al servidor sin el cookie.
4. Si el cookie existe, mira el timestamp asignado al enmascararse para comprobar si aún es válido.
5. Si pasó el tiempo de expiración del mismo lo descarta, caso contrario lo reemplaza por su valor original
6. Si ha pasado un determinado tiempo se realiza un mantenimiento de la base de datos entera removiendo los cookies expirados.
7. El WAF envía el request con los valores originales de los cookies al servidor.

El proceso es completamente transparente para el servidor, pues nunca ve los cookies enmascarados. Para el cliente por otro lado, los valores reales del cookie son imperceptibles e inmodificables.

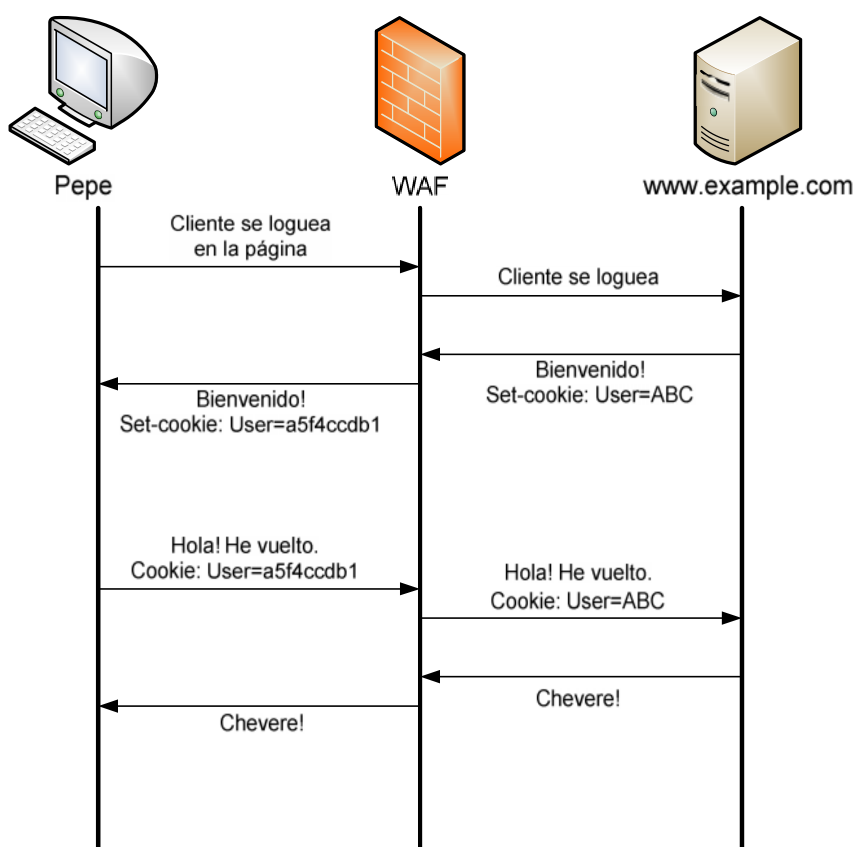

Figura 3.7 Enmascaramiento de cookies

3.5 AUTOAPRENDIZAJE

3.5.1 ANÁLISIS DE TRÁFICO

Una de las principales capacidades de un WAF de nueva generación es la capacidad de generar esquemas de protección para aplicaciones web de manera semi-automática que provean un modelo de seguridad de whitelisting basado en el análisis del tráfico o logs de auditoría generados por el mod_security. Para la generación de este esquema es necesario que el WAF pase por un periodo de tiempo en modo pasivo, haciendo logging sin bloquear ningún tipo de tráfico.

Un esquema de whitelisting está estructurado para cada aplicación como un grupo de recursos, que se refiere a cada una de las páginas del sitio web; a su vez cada recurso puede tener uno o varios comportamientos, que se refieren a la función que prestan al usuario, pues existen recursos con múltiples utilidades, como un script de login que se envíe los datos a sí mismo para luego procesarlos (donde un comportamiento sería presentar al usuario el formulario de login, y otro comportamiento sería recibir los datos de las credenciales y verificar su validez); cada uno de estos comportamientos tienen parámetros

que son básicamente las vías de interacción con el usuario, como los campos de un formulario o incluso cabeceras HTTP.

Llevándolo a un ejemplo más gráfico, la aplicación web hospedada en “ejemplo.com” consta de los recursos “index.php” y “main.php”. El recurso “index.php” tiene dos comportamientos distintos: como pantalla de login de usuarios, y como script para el procesamiento y verificación de credenciales de usuario. En la pantalla de login no tenemos parámetros, mientras que cuando se procesan las credenciales, el mismo recurso (index.php) requiere de un usuario y password. El recurso “main.php”, por otro lado, sólo tiene un comportamiento como portal de inicio del sitio web, que recibe como parámetros un identificador de usuario y una fecha.

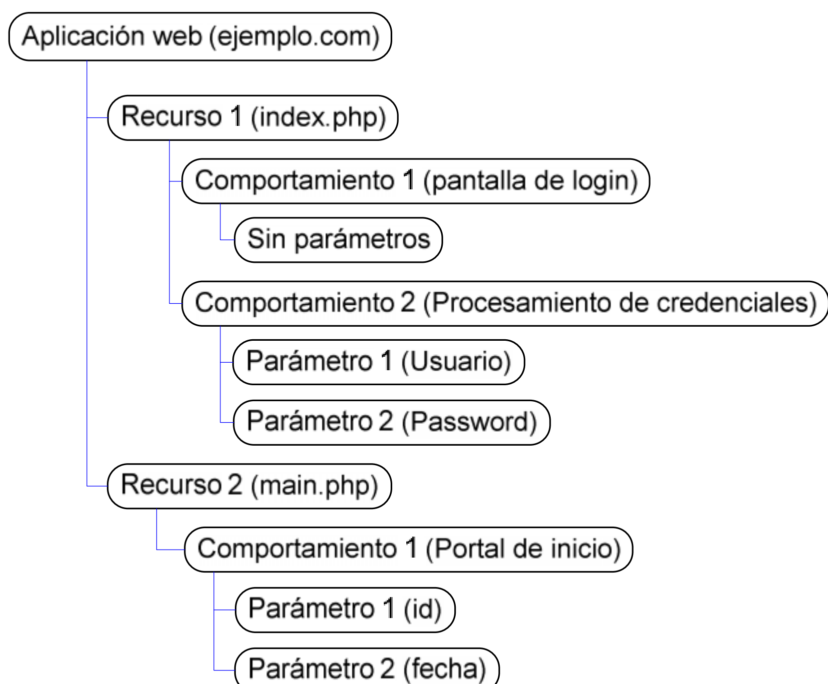

Figura 3.8 Ejemplo de estructura de un sitio

Los elementos de un esquema de whitelisting de una aplicación web cualquiera son los siguientes:

- ***Aplicación web.-*** Una aplicación representa un conjunto de recursos que tienen un mismo fin y los agrupa bajo un mismo nombre.

- ***Recurso.-*** Son todas y cada una de las páginas que conforman una aplicación web. Cada recurso se diferencia por su nombre único, y tiene un valor de "certeza" asignado por el WAF que se

basa en el número de veces que dicho recurso apareció en los logs, pues se supone que mientras más logs existan sobre un determinado recurso, más fiables serán las reglas autogeneradas por el WAF. Cada recurso podrá tener uno o más comportamientos.

- ***Comportamiento.-*** El número de comportamientos de un recurso está basado en la capacidad de brindar diferentes funcionalidades al usuario, como podría ser un script que si no recibe parámetros en el request HTTP, pida al usuario sus credenciales de acceso, pero si recibe dichas credenciales como parámetros, inicie el proceso de login de manera automática. La importancia en la identificación de los diferentes comportamientos de cada recurso radica en el hecho de que cada comportamiento cuenta con diferentes parámetros que permiten a un usuario normal interactuar de maneras distintas, o visto de otra manera, varias posibilidades de ataque al sitio web. Cada comportamiento está definido por un grupo de “precondiciones” que lo diferencian, y tiene al igual que los recursos un grado de “certeza”. Adicionalmente en cada comportamiento se define un rango para el número de parámetros que puede tomar (por ejemplo entre 1 y 3

parámetros), para limitar la cantidad de los mismos que pueden venir en un mismo request, y evitar requests inválidos que consumirían recursos del servidor web. Finalmente se tiene el grupo de parámetros específicos de cada comportamiento.

- ***Parámetros.-*** Los parámetros son los que permiten al usuario interactuar con los recursos. Son todos y cada uno de los datos ingresados por el cliente (esto comprende usuario y navegador) en una petición HTTP, que permiten al usuario manipular el funcionamiento de un determinado recurso. Esto incluye desde los campos de un formulario, hasta cookies o cabeceras HTTP. Cada parámetro tiene un grado de "certeza" tal y como un recurso o comportamiento; tiene un rango de tamaños válidos, que permiten restringir la cantidad de caracteres a ingresarse en ellos para evitar ataques de tipo Buffer Overflow u otros que requieran cadenas largas para su ejecución; tienen una cardinalidad, que se refiere al número de veces que un mismo parámetro puede aparecer en el mismo request (generalmente 1); tienen un tipo, que identifica en una cadena entendible por humanos el contenido "normal" que se puede ingresar en el campo, como pueden ser letras, números, fechas, etc.; y

finalmente una expresión regular que se utilizará como patrón para discernir valores válidos ingresados en el campo.

El script de auto-aprendizaje sigue los siguientes pasos para generar un esquema de seguridad de la página:

1. Se analizan los audit-logs generados por el WAF, tomando en cuenta solo aquellas peticiones que no hayan causado alertas de seguridad.
2. Se procede a detectar todos y cada uno de los recursos, comportamientos y parámetros de la aplicación web.
3. Por cada parámetro, se analiza la lista de valores ingresados en cada request y se los clasifica en grupos de acuerdo a expresiones regulares definidas por el usuario que definen el tipo de dato detectado (entero, alfanumérico, hexadecimal, url, código postal, etc.).
4. Se cuenta el número de datos encontrados para cada tipo y se asigna una ponderación a cada uno dependiendo de la frecuencia de los mismos, donde aquel que tenga más coincidencias obtendrá un valor de 100, y el resto un valor correspondiente porcentual con respecto al máximo.

5. Se descartan todos los tipos de datos que estén por debajo de un umbral porcentual asignado por el usuario para descartar posibles peticiones sospechosas que puedan haber evadido las firmas de detección de ataques por defecto.
6. Se asigna a cada parámetro un patrón que define lo que se podrá ingresar en él, así como su tamaño mínimo y máximo permitidos.

El juego de reglas generado se deberá analizar y adaptar por el administrador para evitar posibles falsos positivos, dado que es altamente restrictivo (pues limita el uso de la página a sólo aquello que se considera válido).

3.5.2 INTEGRACIÓN CON SCANNERS DE VULNERABILIDADES

Una alternativa adicional para el aprendizaje automático de reglas es el uso de los reportes generados por software para scanning de vulnerabilidades. De buenas a primeras esto suena bastante lógico: si tenemos software dedicado a encontrar vulnerabilidades, lo óptimo sería que éste interactuara con nuestros dispositivos de seguridad

reportándoles las fallas detectadas para que estas sean parchadas. Existe, sin embargo, una gran limitación tanto del lado de los scanners de vulnerabilidades como de los dispositivos de red para que esto se dé, y es que los scanners están diseñados de modo que puedan explotar las vulnerabilidades encontradas, mas no solucionarlas, mientras que los dispositivos de seguridad no son capaces de detectar vulnerabilidades en los dispositivos defendidos.

La afectación más directa producto de esto, está en que si bien un scanner de vulnerabilidades podría informar al dispositivo de red de que manera encontró una vulnerabilidad, la firma que podría ser generada por el equipo de seguridad sería demasiado específica como para ser útil. Un pequeño cambio en el vector de ataque, y la firma podría ser fácilmente evadida por un atacante. Existen dos posibilidades para sobrellevar este hecho: la primera sería implementar un scanner de vulnerabilidades al que se le incorpore conocimiento de prevención de ataques, de modo que cada firma que logre explotar una vulnerabilidad contenga también información entendible por el WAF de cómo solucionarla; la otra opción está en hacer que el dispositivo de seguridad reciba un código de vulnerabilidad estandarizado, como por ejemplo la clasificación de amenazas web de WASC (web application security consortium) de

modo que busque en su base de datos de firmas toda aquella relacionada con la vulnerabilidad y las aplique al parámetro vulnerable de manera automática. Este segundo método resulta menos preciso y consume mayor cantidad de recursos, pues se están aplicando posiblemente un número de firmas mucho mayor al necesario, pero sería útil de igual forma.

Al inicio del proyecto se pensó en implementar “conectores” con una aplicación en particular, pero se llegó a la conclusión que crear una interfaz genérica sería de mayor utilidad. Así si un scanner quiere comunicarse con el WAF, lo hará utilizando un formato preestablecido. El formato definido debiera ser lo suficientemente genérico para poderse implementar tanto en cualquier scanner de vulnerabilidades, como en cualquier WAF comercial.

La interfaz propuesta sería a través de archivos XML, los que estarían compuestos de un Vpatch que corresponde a un conjunto de una o varias vulnerabilidades detectadas, así como las medidas recomendadas por el scanner al dispositivo de seguridad. La estructura sería la siguiente:

```
<Vpatch>
      <vuln name="Stored XSS">
            <code>WASC-08</code>
            <site>http://www.example.com/</site>
            <resource>/admin/index.php</resource>
            <param type="arg">user</param>
            <match op="none">null</match>
      <transform>url_decode,remove_spaces,strip_comments
</transform>
            <action>delegate</action>
            <log>audit,fast</log>
            <comment>Ataque de XSS persistente...</comment>
      </vuln>
</Vpatch>
```

Cada vulnerabilidad consta de los siguientes elementos:

- name: Cada vulnerabilidad está identificada por un nombre que sirve de referencia al usuario solamente.
- code: Un código que corresponde a su clasificación según el WASC, clasificación unificada de modo que sea consistente a través de varios sistemas.
- site: El sitio en que la vulnerabilidad fue encontrada, pues un mismo scanning pudo haberse realizado sobre varios sitios.
- resource: El URI del recurso en que se encontró la vulnerabilidad

- param: El campo de la petición a través del cual se pudo explotar la vulnerabilidad mencionada. El parámetro puede ser de uno de los siguientes tipos:
    - req_header: buscar en los encabezados de la petición HTTP.
    - resp_header: buscar en los encabezados de la respuesta HTTP.
    - method: el método HTTP utilizado en la petición.
    - arg: buscar en los argumentos de la petición HTTP.
    - cookie: buscar en las cookies.
    - url: buscar contenido en la URL.
    - global: Buscar contenido en cualquier lugar de los antes mencionados.
    - response: Buscar en el cuerpo de la respuesta HTTP.
- match: permite especificar el operador de comparación y el valor a comparar con el parámetro.
- transform: permite especificar una lista de transformaciones a realizar en el parámetro antes de compararlo con el valor en match.
- action: especifica la acción a ser tomada frente a este patrón de tráfico. Los posibles valores son:
    - drop: descarta el paquete silenciosamente.

- deny: deniega la petición HTTP con código 401.
- accept: Permite el paso del paquete inmediatamente. Útil para implementación de whitelisting.
- pass: Si la regla se dispara, continuar evaluando el resto de reglas. Útil para crear reglas para generación de logs solamente.
- delegate: esta acción sería utilizada cuando el scanner no sepa que acción tomar y prefiera delegar la responsabilidad de la generación de reglas al WAF. El WAF deberá entonces verificar si existe un juego de reglas para aplicar al código enviado en code y ponerlo en funcionamiento.

Nótese que a la fecha no existe ningún scanner o WAF trabajando con este esquema y el modelo se adjunta a esta tesis como una sugerencia solamente de una posible implementación de un framework genérico.

3.6 ALTA DISPONIBILIDAD

Dada la criticidad de la posición del WAF en la red (en línea con el tráfico) se vuelve completamente necesario implementar algún

mecanismo que provea alternativas al paso del tráfico en caso de que el WAF deje de funcionar correctamente. Dado que el WAF se encuentra en un esquema de reverse proxy, no sería posible hacer un bypass del mismo para continuar con el normal funcionamiento de los sitios web protegidos. Esto debido a que el direccionamiento IP de los servidores no correspondería a la red del Gateway de red al que se conectaba el WAF, ya que los servidores web habían sido puestos en una red privada que solo era vista por ellos y por el WAF, mientras que por la otra interfaz el WAF tomaba las IP públicas a las que apuntaban las referencias DNS a los nombres de los sitios.

El esquema a utilizarse requiere de dos WAF para realizarse. Básicamente, lo que se tendrá será un modelo activo-standby, donde uno de los WAF se encuentra frenteando las conexiones de entrada hacia los servidores, mientras que un segundo WAF se encuentra conectado en paralelo y simplemente esperando en caso de que el primer equipo falle. El paso de responsabilidades de un WAF al otro se hace utilizando direcciones IP virtuales que son utilizadas únicamente por el WAF en modo activo.

Ambos WAF se comunican por medio de paquetes que les permiten saber si el otro WAF aún se encuentra activo. Si el WAF en modo standby dejara de detectar estos paquetes (que son enviados de manera periódica), pasaría de manera inmediata a tomar las IP virtuales que utilizaba el WAF activo, volviéndose el nuevo WAF activo.

Nótese que existen IP virtuales tanto en la red de los servidores web, donde el Gateway de todos los servidores apuntará a la IP virtual interna del WAF, y también existen IP virtuales externas que corresponderían a las IP a las que se resuelven los nombres de los sitios web protegidos.

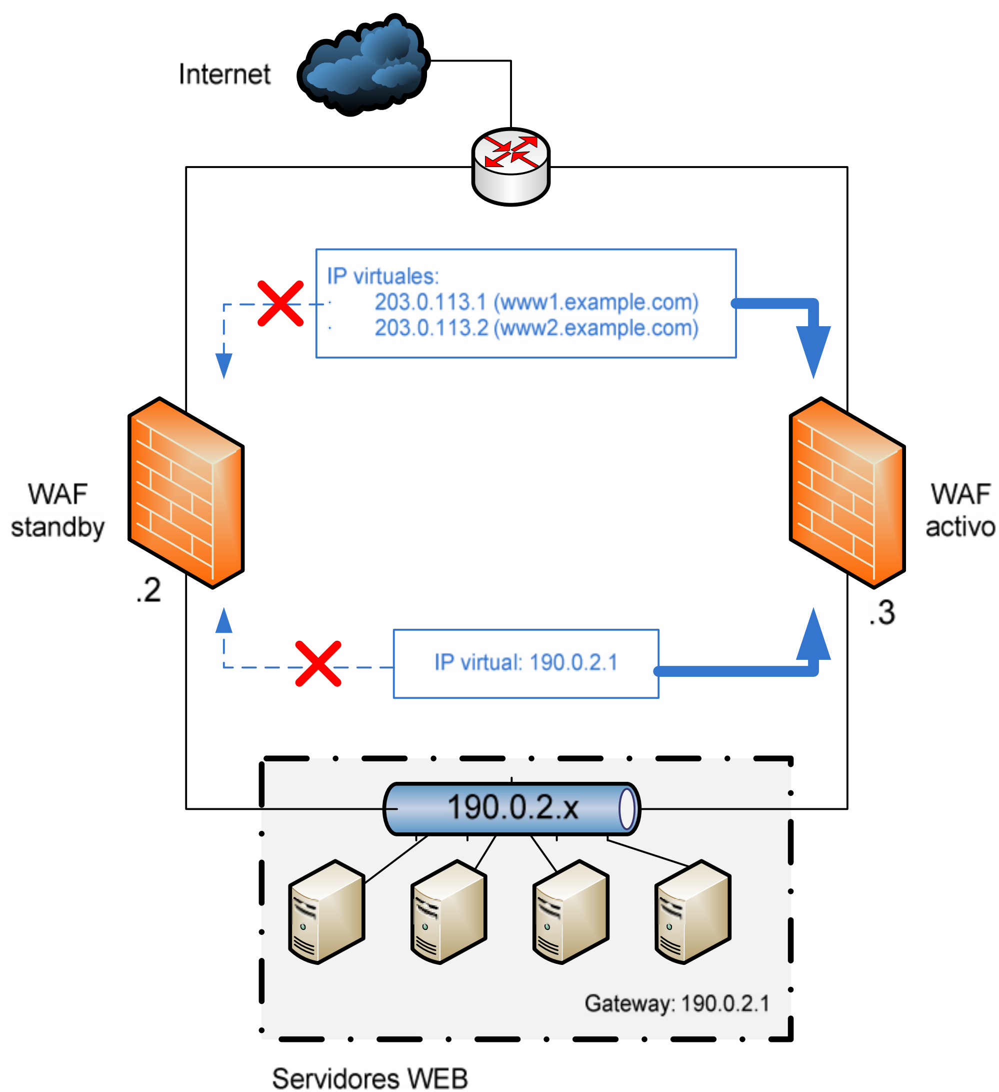

Figura 3.9 Funcionamiento normal del cluster

Las direcciones IP virtuales pueden verse como flotantes entre los dos equipos. Si por alguna razón un equipo dejara de percibir al otro, pasaría a tomar la posta como WAF activo en cuestión de segundos.

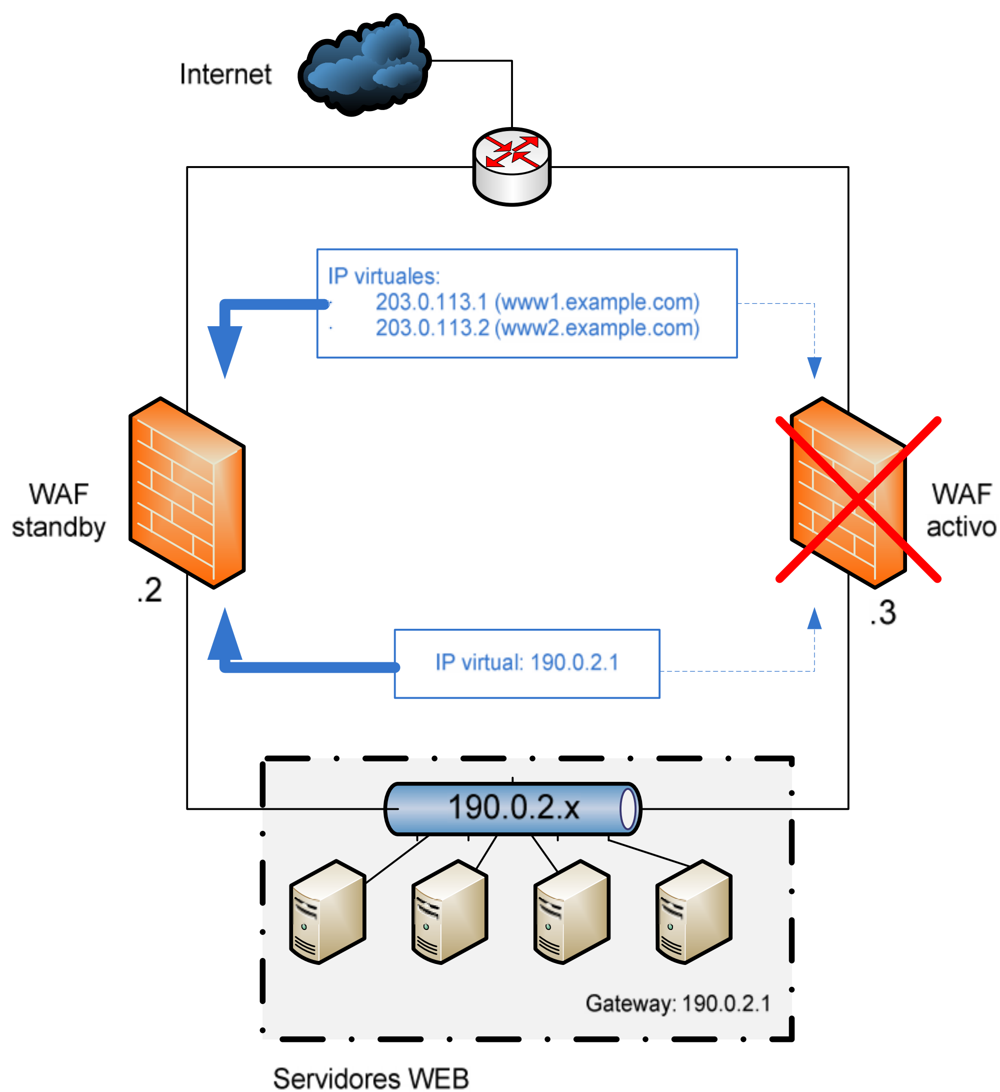

Figura 3.10 Funcionamiento en failover

Para la implementación de este comportamiento se utilizará Heartbeat y Pacemaker, que son herramientas para la creación de clusters en Linux. Heartbeat implementa el protocolo de comunicaciones de bajo nivel para que los WAF puedan detectarse y crear un cluster.

Pacemaker brinda las herramientas necesarias para la administración automatizada del sistema en caso de entrar en modo failover, lo que dicho de otro modo es la interfaz entre los programas puestos en alta disponibilidad y el heartbeat. En resumen, Heartbeat se encarga de que cada WAF sepa de la existencia de otros WAF, mientras que Pacemaker se encarga de cambiar las IP virtuales de WAF cuando se detecta una caída del activo.

CAPITULO 4

INTERFAZ DE MONITOREO

4.1 PROPÓSITO Y ARQUITECTURA DE LA INTERFAZ

La interfaz de monitoreo tiene por objetivo el brindar una manera sencilla de analizar los intentos de ataque detenidos, así como su estructura, permitiendo a un equipo de monitoreo entender lo que está sucediendo a los sitios protegidos en tiempo real. La intención es la de brindar las facilidades para la búsqueda, agrupación y organización de alertas con el objetivo de generar reportes rápidos para ser pasados a los responsables de los sitios, mas no la de proveer una interfaz de configuración del WAF, pues toda la configuración se hará por línea de comandos.

El esquema de la interfaz de monitoreo está basado en Sguil y Squert, ambas herramientas diseñadas en un principio como interfaz de monitoreo de eventos para snort. Sguil provee una infraestructura de agente-servidor que se encarga de recolectar las alertas de uno o varios sensores y ponerlas en una base de datos, de modo que por medio de un cliente estas puedan ser vistas y analizadas; squert es una interfaz web que se conecta a la base generada por Sguil para la presentación a través de una interfaz web de los datos de eventos de seguridad. Ambos componentes tuvieron que ser adaptados para mostrar alertas del WAF.

La adaptación consistió en reemplazar el agente de Sguil, responsable de procesar y colectar los logs de snort, para que entienda y envíe al servidor logs de mod_security. Del lado del cliente web, se tuvo que rediseñar gran parte de las interfaces para adaptarse a las reglas de mod_security, sus alertas y componentes y la auto-categorización de alertas de modo que sea compatible con alertas web en lugar de alertas de IPS (snort).

La arquitectura de sguil permite además que varios agentes reporten a un mismo servidor, de modo que en la misma interfaz de monitoreo

podemos tener varios WAF de manera simultánea, lo que nos permite un control total de alertas de manera centralizada, mejorando las posibilidades de correlacionar eventos entre varios sitios para casos de ataques distribuidos.

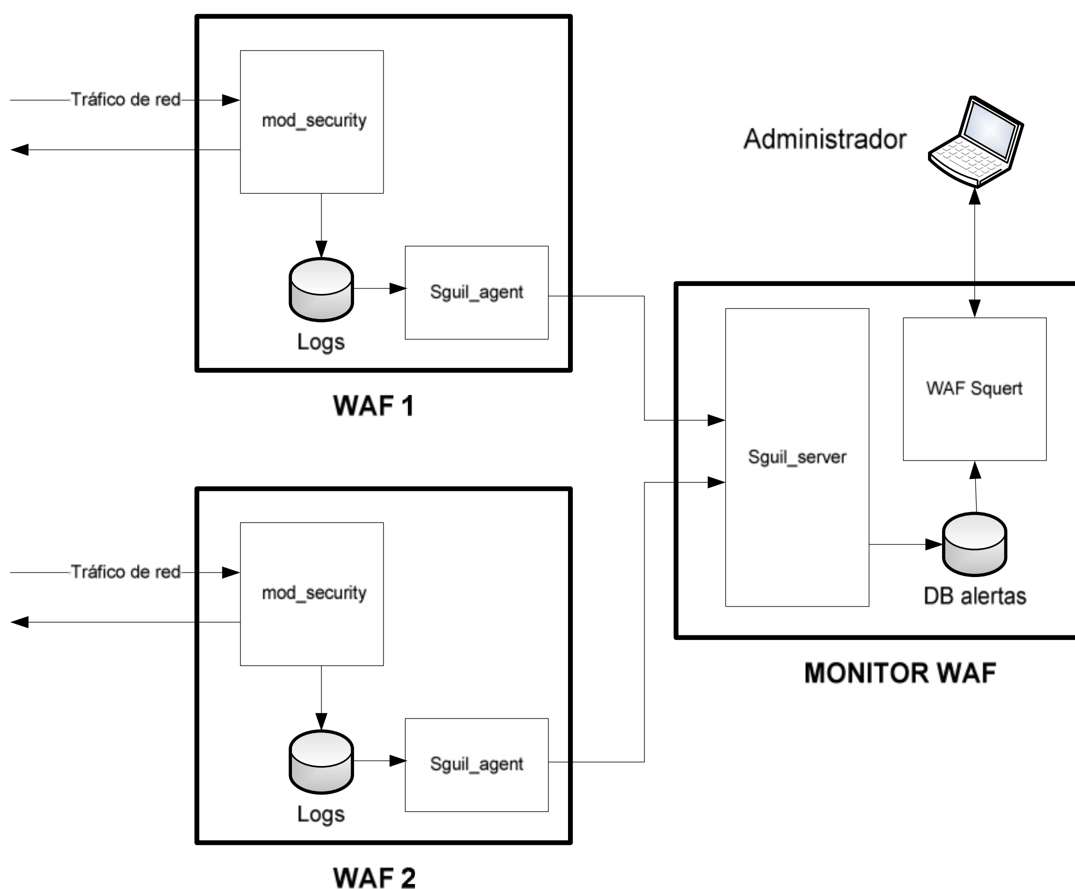

Figura 4.1 Arquitectura de interfaz de monitoreo

4.2 FUNCIONALIDADES DE LA INTERFAZ

El objetivo de una interfaz de monitoreo es el de proveer un nivel de abstracción extra para el análisis de alertas de una manera más general y menos tediosa. Las funcionalidades de la misma deben por tanto proveer facilidades para analizar tendencias y no solo eventos por separado.

Nuestra interfaz se basa en un buscador de eventos que permite realizar el filtrado de los mismos de acuerdo a varios criterios como direcciones IP involucradas en el ataque, puertos TCP de origen o destino, patrones en los nombres de las firmas (por ejemplo 'SQL' para verificar todo ataque que involucre intentos de acceso a la base de datos), o SigID si se sabe el ID exacto de la firma a buscar. Adicionalmente se puede especificar el país de origen del ataque y crear una regla de exclusión que puede involucrar a cualquiera de los parámetros mencionados (Podemos por ejemplo identificar toda alerta de XSS, exceptuando aquellas provenientes de China), así como restringir nuestro análisis a un rango de tiempo entre dos fechas. De este modo se pueden obtener reportes con un control bastante granular que nos permita identificar tendencias en los ataques, como puede ser identificar todos los intentos de ataque desde una

determinada IP o desde un determinado país en una determinada fecha. La figura 4.2 muestra el buscador.

Figura 4.2 Buscador de alertas

Existen tres filtros adicionales que nos permiten filtrar las alertas de un solo sensor (WAF) en caso de tener varios reportando a nuestro colector de logs; podemos también filtrar por “status” que básicamente se refiere a la clasificación de los ataques (XSS, SQLi, OS command injection, etc.); finalmente podemos elegir el tipo de vista retornado que nos permite elegir la manera en la que queremos ver las alertas.

Los tres tipos de vistas existentes para los reportes son:

- Signature: Las alertas son mostradas en grupos de acuerdo a la firma que dispararon.
- Signature, ip: Las alertas son mostradas agrupándolas por firma, y además por direcciones IP fuente y destino.

- Event detail: Nos brinda información detallada de cada alerta, permitiéndonos un análisis de bajo nivel de cada intento de ataque.

Independiente de la presentación elegida, se nos mostrará un mapa de densidades de ataques que nos permitirá observar de manera rápida la concentración de ataques por hora en el rango de tiempo seleccionado, tal y como lo muestra la figura 4.3.

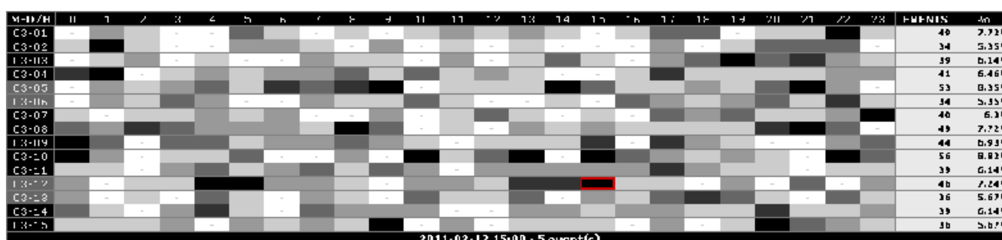

Figura 4.3 Mapa de densidades de eventos

El mapa de densidad de eventos es completamente interactivo y permite dar clic a cualquier hora para restringir las alertas mostradas a solamente esa hora, de modo que se pueda filtrar de manera más precisa la información en pantalla.

También se crearán gráficas que muestran estadísticas de cuantos ataques de cada tipo se dieron en el lapso de tiempo seleccionado, así

como una distribución por hora que muestra la cantidad exacta de alarmas dadas cada hora (exceptuando las horas que no tuvieron alertas), como las mostradas en las figuras 4.4 y 4.5.

Figura 4.4 Eventos agrupados por hora

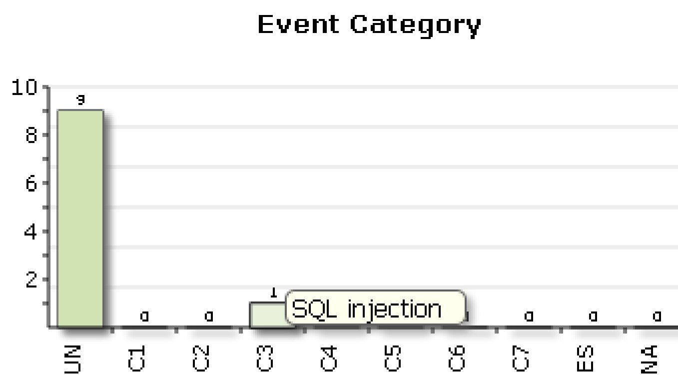

Figura 4.5 Clasificación de eventos por categorías

Adicionalmente se cuenta con la posibilidad de generar un mapa de densidades en cuanto al país de origen del ataque se refiere, permitiéndonos obtener una estadística rápida y comprensible de los lugares que nos atacan más. Esto sería útil para la identificación de fuentes de ataque, y posiblemente para la identificación de razones para los ataques en base a la localización del mismo.

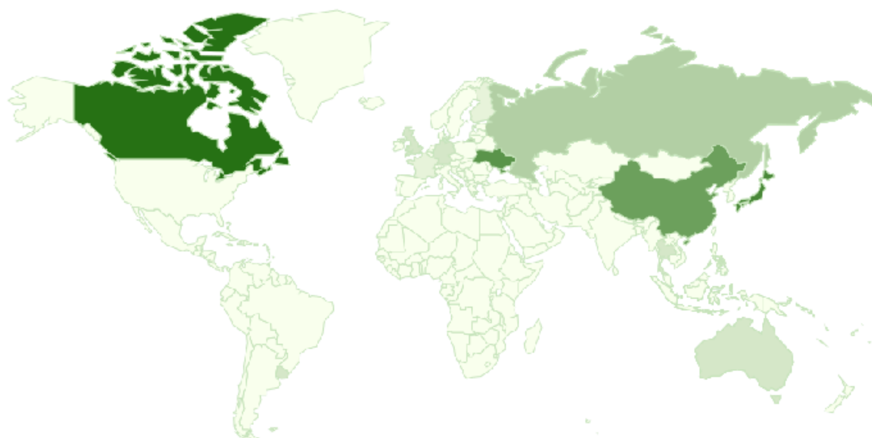

Figura 4.6 Distribución de ataques por país de origen

Se pueden crear diagramas de flujo de ataques para una visualización más sencilla de las alertas, en donde se ilustra claramente la relación entre la firma, atacante y atacado, de modo que pueda ser utilizada para reportes de manera sencilla. Un ejemplo de dichos diagramas se muestra en la figura 4.7

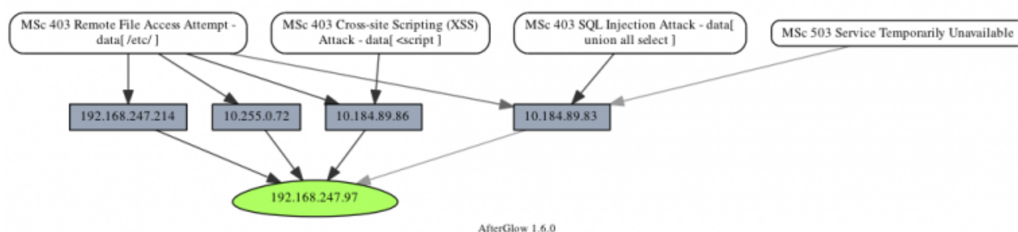

Figura 4.7 Diagrama de ataques

Finalmente, si se requiere un análisis minucioso del ataque, se pueden verificar los detalles de bajo nivel de cada alerta entre los que se incluirán las cabeceras HTTP tanto de la petición como de la respuesta que generó la

alerta, un log que indica la razón de la denegación de la petición, y el cuerpo de la petición si se trata de un POST request. La figura 4.8 muestra la ventana de análisis de eventos y sus diferentes secciones.

INFO	Timestamp	Signature	Signature ID	Sensor ID	Event ID
	11 06 06 13 41:31	MSo 4C3 Remote File Access Attempt data[/etc/]	958700	1	10

TCP/IP	SrcIP	DstIP	SrcPort	DstPort
	11.184.85.83	192.168.247.97	5024	80

REQUEST HEADERS

```
GET /?wtf=ls%20/etc/ HTTP/1.1
Host: inter.net
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept Language: es es,es;q=0.0,en us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0 7,*;q=0 7
Keep-Alive: 115
Connection: keep-alive
Cookie: SECookie2=10.184.89.83.1307385509090310
```

RESPONSE HEADERS

```
HTTP/1.1 403 Forbidden
Content-Length: 202
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content Type: text/html; charset=iso 8859 1
```

TRACE

```
Message. Access denied with code 403 (phase 2). Pattern match "\/etc\/" at ARGS.wtf. [file "/etc/htt
pd/rules/base_rules/modsecurity_crs_40_generic_attacks.conf"] [line "220"] [id "958700"] [rev "2.0.1
0"] [msg "Remote File Access Attempt"] [data "/etc/"] [severity "CRITICAL"] [tag "WEB_ATTACK/FILE_IN
JECTION"] [tag "WASCTC/WASC-33"] [tag "OWASP_TOP_10/A4"] [tag "PCI/6.5.4"]
Action: Intercepted (phase 2)
Apache Handler: proxy server
Stopwatch: 1307385661304816 837 (333 619 -)
Producer: ModSecurity for Apache/2.5.13 (http://www.modsecurity.org/); core ruleset/2.0.10.
Server: Apache
```

Figura 4.8 Inspección a fondo de eventos

CAPITULO 5

PRUEBAS DE RENDIMIENTO

5.1 ESCENARIO DE PRUEBAS

Para las pruebas de rendimiento se montó un pequeño laboratorio que consta de los siguientes componentes:

- 1 WAF
- 1 servidor web con varios sitios web vulnerables preparados específicamente para las pruebas
- 1 equipo actuando como cliente, quien realizará consultas al servidor web

Las características del WAF son las siguientes:

- CPU: Intel(R) Pentium(R) CPU E5300 @ 2.60GHz

- RAM: 2GB
- SO: ArchLinux x64 (kernel 2.6.37-ARCH)

La topología de red utilizada fue la siguiente:

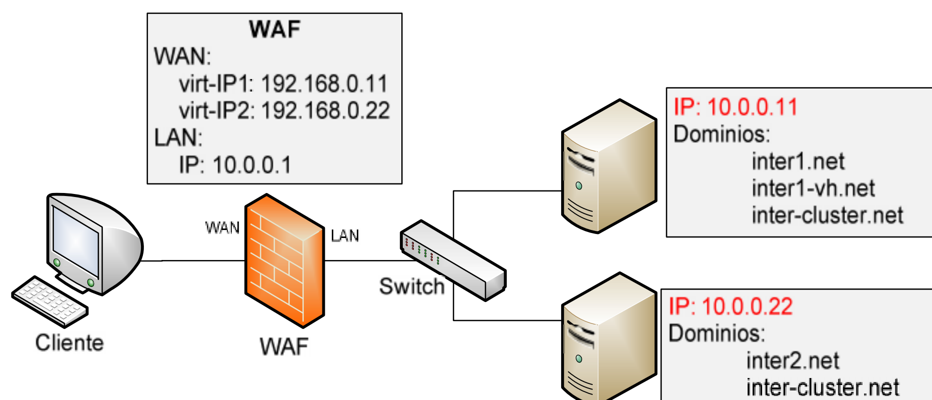

Figura 5.1 Escenario de pruebas

Nótese que físicamente sólo existe un servidor web, pero el mismo contiene dos direcciones IP para la emulación de varios servidores. Los dominios alojados en cada servidor son completamente independientes a excepción de inter-cluster.net que será utilizado para probar la funcionalidad de balanceo de carga.

5.2 ANCHO DE BANDA Y SESIONES CONCURRENTES

Esta primera prueba tiene por objetivo medir la cantidad de información que el WAF será capaz de procesar por unidad de tiempo. Dado que no se quiere que parámetros de latencia afecten el rendimiento de las conexiones, el tipo de contenido servido al cliente será puramente estático. Las variables a medir serán el ancho de banda efectivo en Kilobits por segundo, utilización de CPU y memoria, y variabilidad de la latencia dependiendo de la carga del dispositivo. Los parámetros a variar serán el tamaño de las peticiones, cantidad de peticiones por segundo y ancho de banda utilizado dependiendo de la prueba.

Primeramente, las pruebas de ancho de banda se las realizó tomando en cuenta la suma del ancho de subida con el de bajada, pues ambas partes son procesadas por el WAF. Las páginas servidas fueron de tamaños variados y tomadas de sitios visitados frecuentemente por un usuario común.

La primera prueba se la realizó sin utilizar la capacidad de keepalive del WAF, lo que quiere decir que por cada conexión TCP el cliente puede realizar una sola transacción HTTP. El resultado obtenido se representa en las figuras 5.2 ,5.3 y 5.4 para el ancho de banda, la cantidad de sesiones concurrentes y el uso de recursos del servidor respectivamente, evaluadas en base a la cantidad de peticiones por segundo enviadas al servidor.

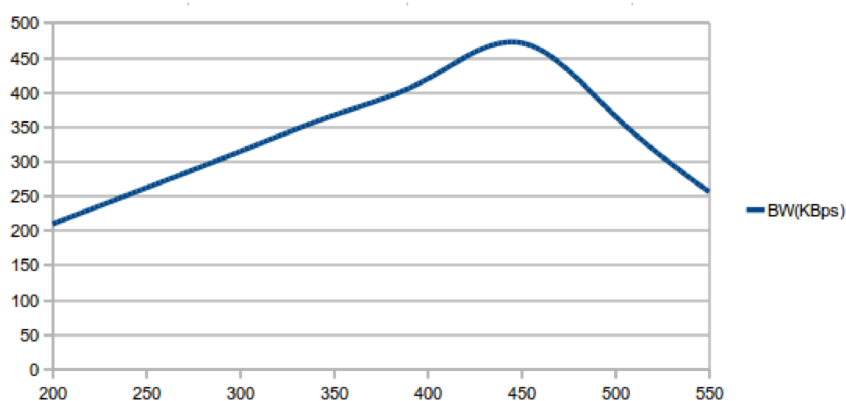

Figura 5.2 Ancho de banda alcanzado sin keepalives

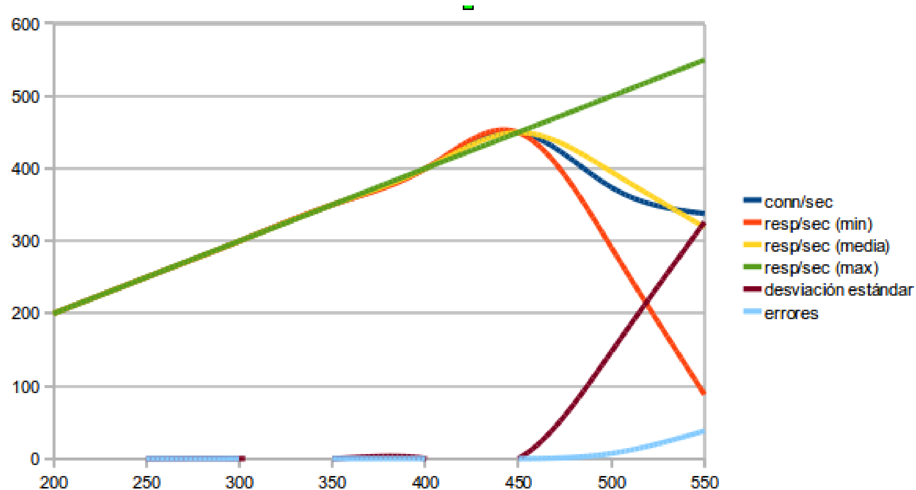

Figura 5.3 Sesiones concurrentes sin keepalives

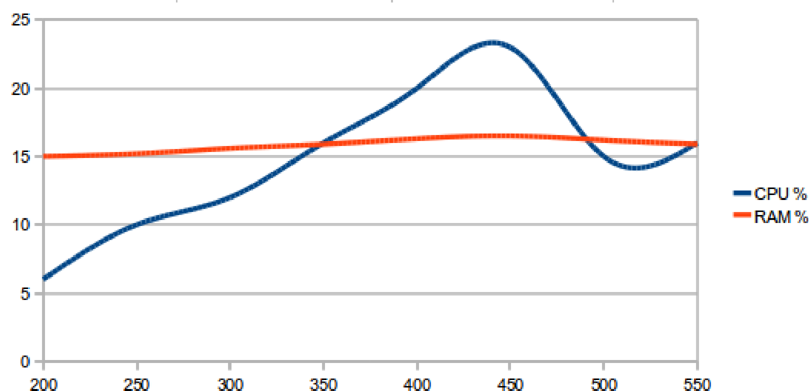

Figura 5.4 Uso de CPU y RAM sin keepalives

De las gráficas se puede notar claramente que existe un punto de quiebre alrededor de las 450 peticiones por segundo, momento a partir del cual el servidor se vuelve inestable. El ancho de banda máximo alcanzado es de 476 KBytes por segundo, equivalente a 3.8Mbps. La limitación está en la implementación de Apache, que impone por software un tope al número de conexiones que puede procesar de manera paralela el servidor. Es por esto que no se notan saturaciones ni a nivel de CPU ni de RAM, sino que más bien es la cantidad de sesiones establecidas en paralelo las que hacen que el servidor se vuelva inestable. Este límite puede ajustarse, pero el subirlo demasiado podría provocar que Apache agote los recursos del servidor.

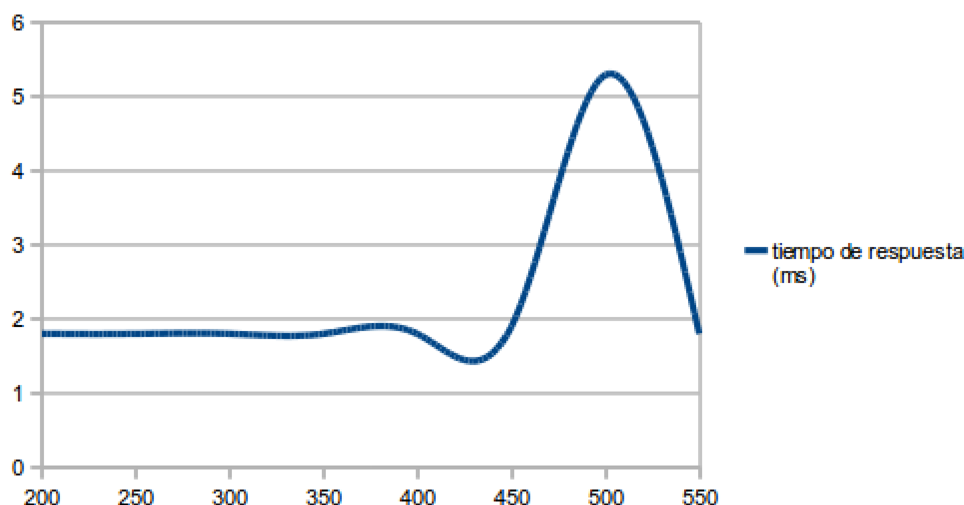

Figura 5.5 Latencia por transacción HTTP sin keepalives

En cuanto a latencia, el WAF muestra un comportamiento bastante estable hasta llegar al punto de quiebre, manteniendo un valor casi constante de 1.8ms por transacción hasta llegar a las 500 peticiones por segundo, momento en el cual la latencia se multiplica por un factor de 4. Esto se muestra en la Figura 5.5.

La siguiente prueba se realizó utilizando el mecanismo de keepalives proporcionado por Apache, lo que permite aumentar en gran manera la capacidad del servidor, ya que si bien el número de conexiones realizadas hacia el servidor se encuentra limitada, no lo están el número de peticiones por segundo. Los resultados de esta prueba fueron mucho más satisfactorios:

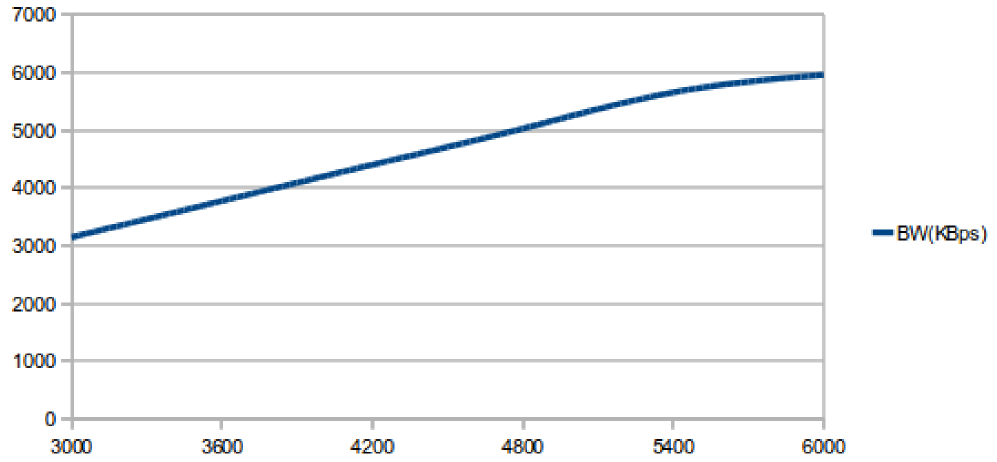

Figura 5.6 Ancho de Banda alcanzado con keepalives

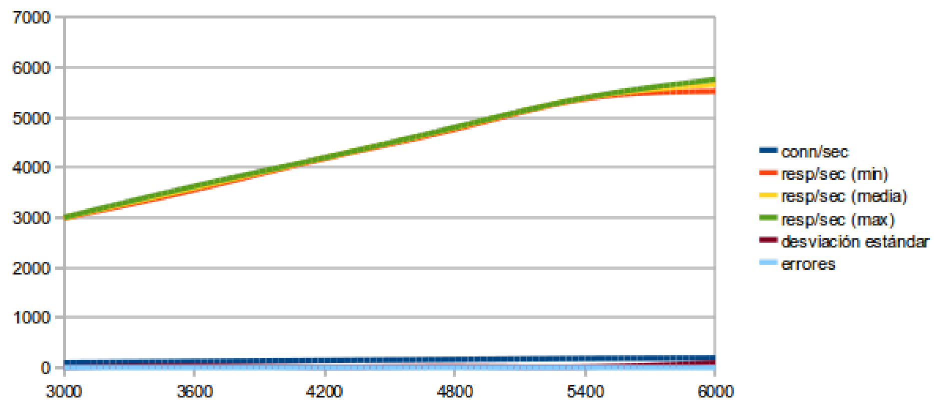

Figura 5.7 Sesiones concurrentes con keepalives

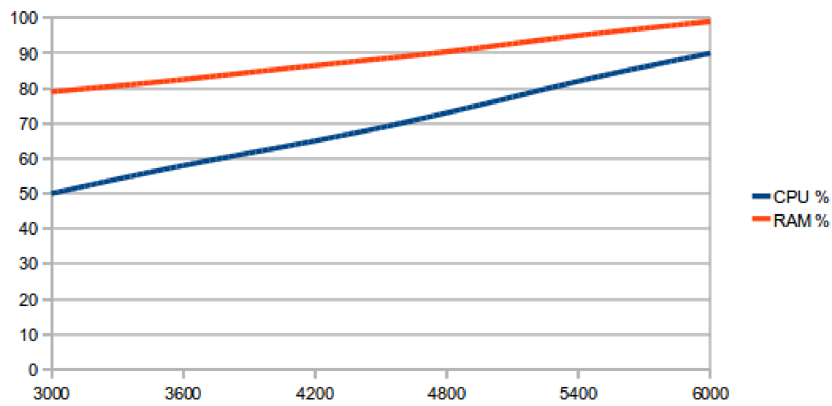

Figura 5.8 Uso de CPU y RAM con keepalives

Para la prueba con keepalives se enviaron 30 peticiones HTTP por conexión TCP abierta. Como se observa claramente, el rendimiento alcanzado es mayor en un factor de más o menos 10. El punto de quiebre se encuentra ahora a unas 5500 peticiones HTTP por segundo, momento en que el ancho de banda llegaba a 5732 Kilobytes por segundo, equivalentes a 45.9 Mbps de tráfico, cantidad suficiente para alojar entre 5 y 10 servidores web de carga media. El factor determinante del punto de quiebre en este caso fue la sobreutilización de recursos del sistema.

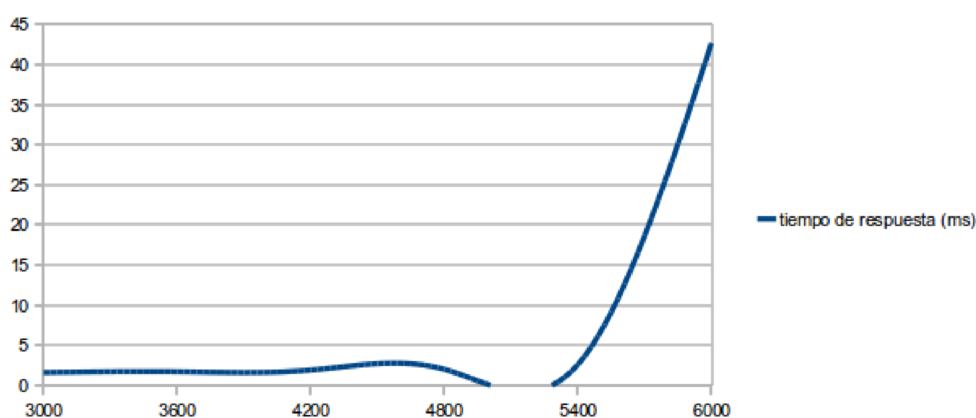

Figura 5.9 Latencia por transacción HTTP con keepalives

CONCLUSIONES

1. La creación de un appliance de bajo costo para la protección de sitios web enfocado a PyMEs es posible bajo el esquema de implementación mostrado, ya que no se requiere ningún tipo de hardware especializado, y el software es completamente open-source.

2. El rendimiento alcanzado nos permite proteger webhostings enteros de tamaño medio a alto, sin insertar una latencia significativa en el procesamiento de los requerimientos.

3. El esquema de auto-aprendizaje implementado es capaz de crear un esquema de seguridad positivo de gran efectividad, restringiendo las posibilidades del usuario en el sitio web a las mínimas necesarias, sin

causar un impacto notable en la experiencia de navegación de un usuario común.

4. La combinación de ambos esquemas de seguridad, tanto el positivo como el negativo, nos permiten brindar una protección de calidad a la gran mayoría de sitios de manera casi automática.

5. La solución creada no requiere de gran conocimiento del protocolo HTTP para ser puesta en funcionamiento, expandiendo así el alcance de soluciones de este tipo a una cantidad aún mayor de potenciales usuarios.

RECOMENDACIONES

1. Dada la naturaleza del WAF implementado, este podría utilizarse para brindar servicios de SaaS (Security as a Service), brindando protección a sitios “desde la nube”.
2. El enmascaramiento de cookies puede mejorarse aún más haciendo un binding de cada sesión a la IP en que se originó, evitando la posibilidad de realizar ataques de robo de cookies.
3. Se puede lograr mayor eficacia en el SSL offloading por medio de aceleradores SSL en hardware, descargando al CPU de la carga aumentada por cifrado de conexiones.

4. La capacidad de sesiones del dispositivo puede mejorarse utilizando una base distinta a Apache, como un reverse proxy basado en Nginx, aumentando el rendimiento de manera significativa.

ANEXOS

A.1 REFERENCIA DE MODSECURITY

Variables Utilizables

ARGS	PERF_SREAD	SCRIPT_MODE
ARGS_COMBINED_SIZE	PERF_SWRITE	SCRIPT_UID
ARGS_GET	QUERY_STRING	SCRIPT_USERNAME
ARGS_GET_NAMES	REMOTE_ADDR	SERVER_ADDR
ARGS_NAMES	REMOTE_HOST	SERVER_NAME
ARGS_POST	REMOTE_PORT	SERVER_PORT
ARGS_POST_NAMES	REMOTE_USER	SESSION
AUTH_TYPE	REQBODY_ERROR	SESSIONID
DURATION	REQBODY_ERROR_MSG	STREAM_INPUT_BODY
ENV	REQBODY_PROCESSOR	STREAM_OUTPUT_BODY
FILES	REQUEST_BASENAME	TIME
FILES_COMBINED_SIZE	REQUEST_BODY	TIME_DAY
FILES_NAMES	REQUEST_BODY_LENGTH	TIME_EPOCH
FILES_SIZES	REQUEST_COOKIES	TIME_HOUR
FILES_TMPNAMES	REQUEST_COOKIES_NAMES	TIME_MIN
GEO	REQUEST_FILENAME	TIME_MON
HIGHEST_SEVERITY	REQUEST_HEADERS	TIME_SEC
INBOUND_ERROR_DATA	REQUEST_HEADERS_NAMES	TIME_WDAY
MATCHED_VAR	REQUEST_LINE	TIME_YEAR
MATCHED_VARS	REQUEST_METHOD	TX
MATCHED_VAR_NAME	REQUEST_PROTOCOL	UNIQUE_ID
MATCHED_VARS_NAMES	REQUEST_URI	URLENCODED_ERROR
MODSEC_BUILD	REQUEST_URI_RAW	USERID
MULTIPART_CRLF_LF_LINES	RESPONSE_BODY	WEBAPPID
MULTIPART_STRICT_ERROR	RESPONSE_CONTENT_LENGTH	WEBSERVER_ERROR_LOG
MULTIPART_UNMATCHED_BOUNDARY	RESPONSE_CONTENT_TYPE	XML
PATH_INFO	RESPONSE_HEADERS	
PERF_COMBINED0	RESPONSE_HEADERS_NAMES	
PERF_GC	RESPONSE_PROTOCOL	
PERF_LOGGING	RESPONSE_STATUS	
PERF_PHASE1	RULE	
PERF_PHASE2	SCRIPT_BASENAME	
PERF_PHASE3	SCRIPT_FILENAME	
PERF_PHASE4	SCRIPT_GID	

PERF_PHASE5	SCRIPT_GROUPNAME	

Transformaciones Disponibles

base64Decode	length	replaceComments
base64DecodeExt	lowercase	replaceNulls
base64Encode	md5	urlDecode
cmdLine	none	urlDecodeUni
compressWhitespace	normalisePath	urlEncode
cssDecode	normalisePathWin	sha1
escapeSeqDecode	parityEven7bit	trimLeft
hexDecode	parityOdd7bit	trimRight
hexEncode	parityZero7bit	trim
htmlEntityDecode	removeNulls	
jsDecode	removeWhitespace	

Acciones Disponibles

allow	logdata	sanitiseRequestHeader
append	msg	sanitiseResponseHeader
auditlog	multiMatch	severity
block	noauditlog	setuid
capture	nolog	setsid
chain	pass	setenv
ctl	pause	setvar
deny	phase	skip
deprecatevar	prepend	skipAfter
drop	proxy	status
exec	redirect	t
expirevar	rev	tag
id	sanitiseArg	xmlns
initcol	sanitiseMatched	
log	sanitiseMatchedBytes	

Operadores Disponibles

beginsWith	le	validateByteRange
contains	lt	validateDTD
endsWith	pm	validateSchema
eq	pmf	validateUrlEncoding
ge	pmFromFile	validateUtf8Encoding
geoLookup	rbl	verifyCC
gsbLookup	rsub	verifyCPF
gt	rx	verifySSN
inspectFile	streq	within
ipMatch	strmatch	

BIBLIOGRAFÍA

1. Apache Foundation – Documentación de Apache HTTPD 2.2 – http://httpd.apache.org/docs/2.2/ – 2011
2. Ristic, Ivan – Apache Security - Feisty Duck Ltd – 2005
3. OWASP – OWASP Top 10 Project – https://www.owasp.org/index.php/Top_10_2010 – 2010
4. Kew, Nick – The Apache Module Book – Prentice Hall - 2007
5. Trustwave – ModSecurity Reference Manual – http://sourceforge.net/apps/mediawiki/mod-security/index.php?title=Reference_Manual – 2011
6. Magnus, Mischel - ModSecurity 2.5 – Packt Publishing – 2009
7. Ristic, I. / Shezaf O. – Enough with Default Allow in Web Applications – http://blog.modsecurity.org/files/Breach_Security_Labs-Enough_with_Default_Allow.pdf – 2008