Implementación de un Web Application Firewall Basado en mod_security

Flament Georges, Vera Johanna, Aranda Alfonso Ing.
Facultad de Ingeniería Eléctrica y Computación
Escuela Superior Politécnica del Litoral (ESPOL)
Campus Gustavo Galindo, Km 30.5 vía Perimetral
Apartado 09-01-5863. Guayaquil-Ecuador
benji0256@telconet.net, jvera@telconet.net, aaranda@telconet.net

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.

Palabras Claves: *Seguridad Web, Web Application Firewall, mod_security.*

Abstract

Given that many server intrusions are done via web nowadays, it is important to secure web servers against malicious users. It becomes imperative to have an extra security layer to protect us our information.

There are several solutions provided by many vendor, but all of them at high prices. The present paper will show that it is possible to implement a low-cost Web Application Firewall, making it possible for anyone to access this kind of technologies. To achieve our objective, the project will be based purely on Open Source software. The final product will have the same features as existing products, making it competitive.

The final product will be a complete appliance, including hardware, software and a management interface in a bundle that can be used by any network administrator.

Keywords*: Web Security, Web Application Firewall, mod_security.*

1. Introducción

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.

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.

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.

4. Marco Teórico

4.1. Firewall 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.

4.2. Apache HTTPD

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.

4.3. mod_security

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.

5. Diseño de la Solución

5.1. 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.

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.

5.2. Optimización de Tráfico Web

Dado que estamos trabajando con un reverse proxy, resulta conveniente implementar funcionalidades extra que permitan ampliar el campo de aplicación del WAF. Estas funcionalidades, aun no siendo propias de un WAF, permitirán mejorar el rendimiento del mismo y de los sitios web protegidos.

La características implementadas son las siguientes:

- Caching de páginas web: Permite guardar copias de páginas estáticas en el WAF de modo que no tengan que ser solicitadas varias veces a los servidores en el backend.
- Balanceo de Carga: Permite distribuir el tráfico entre varias réplicas de un mismo servidor, conocidas como granja de servidores, de modo que se mejore el rendimiento del sitio web utilizando varios servidores.
- Offloading SSL: Se puede descargar el CPU del servidor protegido, delegando al WAF la función de cifrar la conexión utilizando SSL. Esto tiene dos ventajas bastante claras: primeramente se quita la carga de procesamiento que agrega la capa de cifrado de la comunicación, permitiendo al servidor web dedicarse a lo que realmente le corresponde, que es procesar los datos del sitio web; por otro lado si un servidor web no soporta el uso de SSL para cifrado de conexiones, el WAF puede proveerlo de manera transparente, sin necesidad de reconfigurar el servidor de backend.

5.3. Manejo 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.

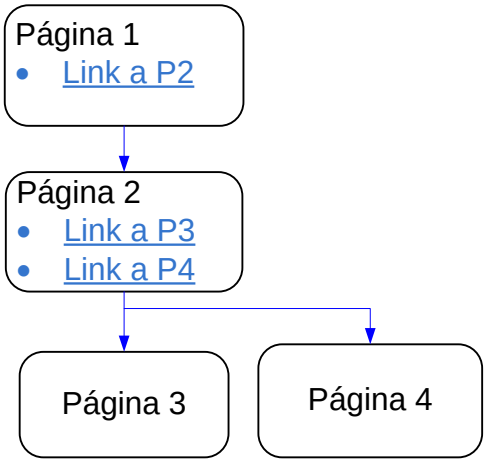

Figura 1. Árbol de Flujo de un Sitio Web

5.4. 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 remplazar 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 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.

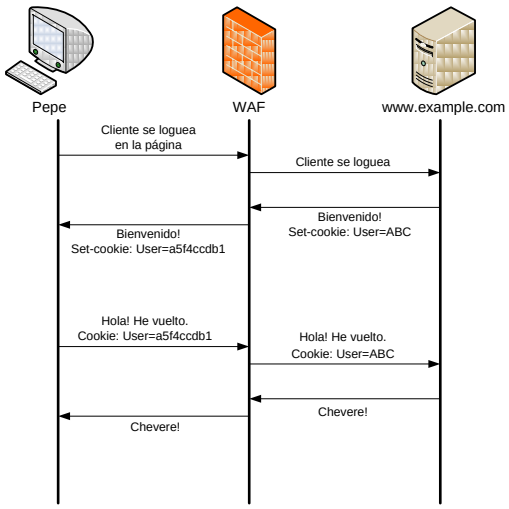

Figura 2. Enmascaramiento de Cookies

5.5. Auto-aprendizaje

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.

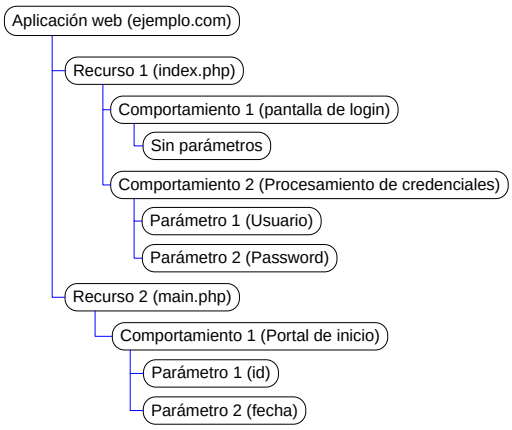

Figura 3. Estructura de un Sitio Web

5.6. 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.

5.7. Interfaz de Monitoreo

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.

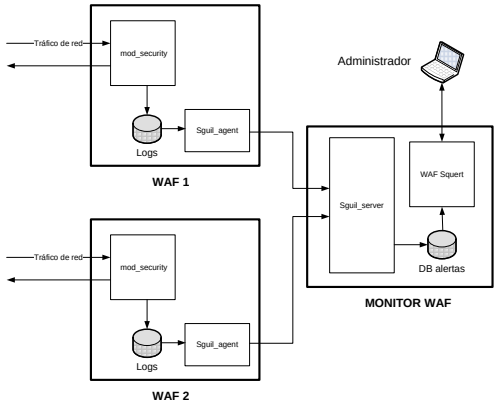

Figura 4. Diagrama de Bloques Sguil

6. Pruebas de Rendimiento

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)

Se activó la función de keepalives para mejorar el rendimiento del WAF, permitiendo realizar varias peticiones HTTP en una misma conexión.

A continuación las gráficas de resultados concernientes a anchos de banda y cantidad de conexiones concurrentes soportadas:

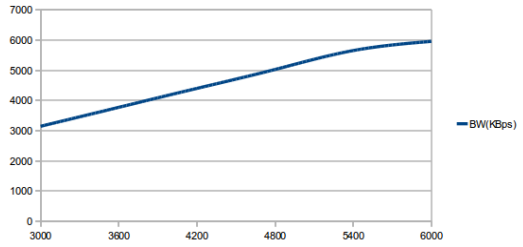

Figura 5. Ancho de Banda Alcanzado

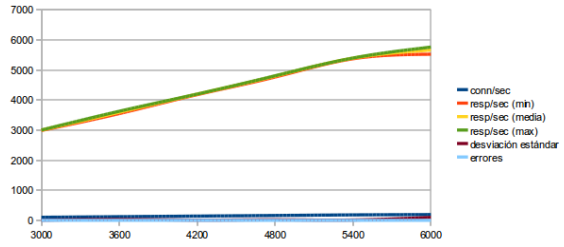

Figura 6. Conexiones Concurrentes Alcanzadas

Para la prueba con keepalives se enviaron 30 peticiones HTTP por conexión TCP abierta. El punto de quiebre se encuentra 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, pues el uso de CPU bordeaba el 100%.

Nótese que la latencia introducida por el WAF resulta bastante estable y alrededor de los 2 ms. Al llegar al punto de quiebre, se dispara de manera casi lineal, como se muestra en la Figura 7.

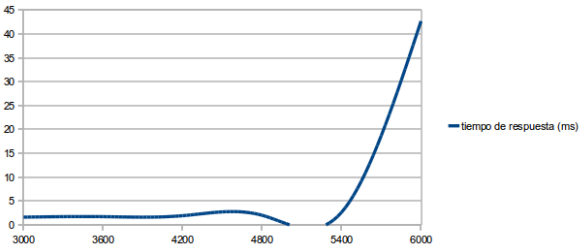

Figura 7. Latencia vs. Peticiones por Segundo

7. 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.

14. Referencias

[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.