Proyecto Fin de Grado
Ingeniería de Telecomunicaciones

Implementación y medida del rendimiento de un Firewall para aplicaciones Web (WAF) en un balanceador de carga

Autor: Álvaro Cano Blanco
Tutor: Antonio Jesús Sierra Collado

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2013

Proyecto Fin de Grado
Ingeniería de Telecomunicación

Implementación y medida del rendimiento de un Firewall para aplicaciones Web (WAF) en un balanceador de carga

Autor:
Álvaro Cano Blanco

Tutor:
Antonio Jesús Sierra Collado
Profesor

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Implementación y medida del rendimiento de un Firewall para aplicaciones Web (WAF) en un balanceador de carga

Autor: Álvaro Cano Blanco

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

Agradecimientos

A mis amigos, a mi familia, a mis compañeros y a mis profesores. En definitiva a todos aquellos de los que alguna vez aprendí algo.

Álvaro Cano Blanco

Sevilla, 2018

Resumen

Debido la masificación de clientes en Internet, y a los inevitables cortes que un servicio puede sufrir por su propia naturaleza, la tecnología está cambiando de paradigma. Los nuevos conceptos tecnológicos como son *Big data*, *machine learning* o *data science*, se basan en aplicaciones distribuidas con las que paliar las saturación de usuarios y dotar a la aplicación de alta disponibilidad, con la que se garantiza el acceso al servicio y de calidad.

Este paradigma ha sido muy bien adoptado por las empresas, que ven en él una forma de ganarse la confianza de clientes, ya que éstos perciben un servicio de calidad, la empresa dispone de alta disponibilidad en aplicaciones críticas y se abaratan costes, debido a que la infraestructura escala fácilmente en función a la demanda de clientes.

El rol que tiene el balanceador de carga como elemento centralizador del tráfico que circula hacia las aplicaciones distribuidas, lo hace ideal como punto en el cual aplicarles las políticas de seguridad.

Esto presenta varias mejoras en el mantenimiento y rendimiento de la aplicación, como son: que la gestión de la seguridad se hace en un único nodo; se valida la información que reciben las aplicaciones, haciendo posible que la comunicación con ésta pueda viajar sin cifrar; y se aprovechan algunas de las operaciones del proxy Web que son compartidas con el WAF (Web Application Firewall) con lo que se elimina latencia.

Con este proyecto se ha dotado a un balanceador de carga de capacidades WAF (Web Application Firewall), utilizando una librería de inspección de contenido que permite configurar reglas con las que proteger los servidores frente a amenazas y vulnerabilidades. El desarrollo se ha llevado a cabo teniendo en cuenta que el rendimiento fuese lo más óptimo posible y que el servicio de balanceo no sufra cortes durante la configuración de las políticas de seguridad, dado su criticidad.

Para abstraerse de la librería y darle al operador de seguridad una herramienta fácil e intuitiva, se ha creado una API HTTP con la que pueda aplicar configuraciones sin tener que conocer la semántica de reglas usada por la librería elegida. Esta API puede ser utilizada para diversas tareas, como integrarse en otros sistemas, por ejemplo, un panel de control; o automatizar tareas a través de *scripting*.

Abstract

Due to the mass of clients on the internet, and the inevitable breaks in service that an application can experience because of its very nature, technology is changing its paradigm. New technical concepts like Big Data, Machine Learning and Data Science, are based on distributed applications that solve the user saturation and gives the system high availability, and the guarantee of accessing a quality service.

This paradigm has been widely adopted by companies, which are using these new techniques to gain the clients' confidence. Therefore the clients can receive quality services, the companies obtain better infrastructures in their critical applications and they save waste.

The load balancer role, as a centralizing element of traffic to the distributed application, makes it an ideal point to apply the network security and ensure its veracity.

This introduces some improvements in the maintenance and performance of the applications, they are: the security management is done in a single node; the information that the applications receive is validated, making communication between the load balancer and the processing nodes possible without a cipher; and it is possible to take advantage of some proxy operations that are shared with the WAF (Web Application Firewall), removing latency.

In this project, a load balancer has been improved with WAF feature, using a content inspection library that allows setting security rules that help protect the Web servers from security threats and vulnerabilities. The development has been done, bearing in mind, the performance and not disturbing the load balancing services while the security policies are being configured, due the load balancer critical role for the provided service.

An HTTP API has been developed with the purpose of abstracting the WAF library and providing the security administrators a tool which will let configure the system easily. The API compiles JSON objects to complex SecLang directives. The API can be used for several tasks like: integrating the system in another solution, for example, a graphic control panel; or automating jobs through scripting languages.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Nos ha tocado vivir un mundo basado en la tecnología, y, ¿qué mejor forma de entenderlo, que a través de sus comunicaciones?

1.1 Presentación

El crecimiento de los usuarios y dispositivos en Internet, está degenerando en problemas de acceso a los servicios que ésta alberga. Constantemente nos encontramos con servicios de baja calidad, debido a la saturación de clientes o incluso cortes de servicios, que pueden ir desde minutos hasta semanas.

Esta degeneración de los servicios puede ser debido a varios factores como:

- Límite de recursos en la aplicación.
- Problemas en la infraestructura.
- Operaciones de mantenimiento.
- Explotación de vulnerabilidades.
- Ataques de denegación de servicio, DoS.

Ante esto, las empresas empiezan a demandar soluciones tecnológicas con aplicaciones distribuidas que permitan un rápido escalado de la infraestructura, a la vez, que puedan dar un servicio más robusto que no sea amenazado con cortes de servicio o que disminuya su calidad.

En la implementación de estas tecnologías empiezan a aparecer nombres como *Big Data*, *High Availability*, o *Data Science*, en los que el balanceador de carga es el elemento que se encarga de centralizar el tráfico y orquestarlo de forma óptima entre los distintos nodos de procesamiento de datos.

De cara a la seguridad, actualmente, existe una demanda continua de sistemas que ofrezcan propiedades capaces de detectar y prevenir los distintos tipos de ataques. Entre los ataques más comunes y eficaces, se encuentran los que van dirigidos a las aplicaciones Web (SQL Injection, DoS, CSS, fuerza bruta…) y ante los cuales, el WAF, es la herramienta usada como elemento de protección.

El WAF es un elemento flexible, que puede configurarse en función de la aplicación que esté protegiendo. Es un sistema que aplica una política (bloquear o continuar) a una transacción HTTP cuando se cumple un determinado patrón. La asociación de estos patrones y políticas se hace mediantes reglas de configuración que serán comprobadas en cada trama HTTP que se dirija o salga de la aplicación.

Debido a que el WAF necesita desempaquetar las tramas HTTP para buscar patrones, se presenta una oportunidad de mejora del sistema al aprovechar que el LB (Load Balancer) también necesita realizar esta tarea.

1.2 Motivación

Debido a la complejidad que añade la seguridad a las infraestructuras y los sistemas, el interés de este proyecto radica en dar a los operadores de seguridad una herramienta útil e intuitiva que permita aplicar configuraciones

de seguridad en el balanceador de carga. Sin tener que mantener reglas y directivas de seguridad en cada uno de los nodos de procesameinto.

En este proyecto se dará un primer alcance para esta herramienta. Éste es que puedan ser examinados los campos HTTP más frecuentes y se les puedan aplicar condiciones que repercutan en aceptación o denegación de la petición entrante a la aplicación Web.

1.3 Objetivos

El objetivo de este proyecto es dar capacidades de WAF (Web Application Firewall) a un balanceador de carga, con el sentido de que el balanceador pueda aceptar o denegar peticiones de clientes si detecta alguna amenaza.

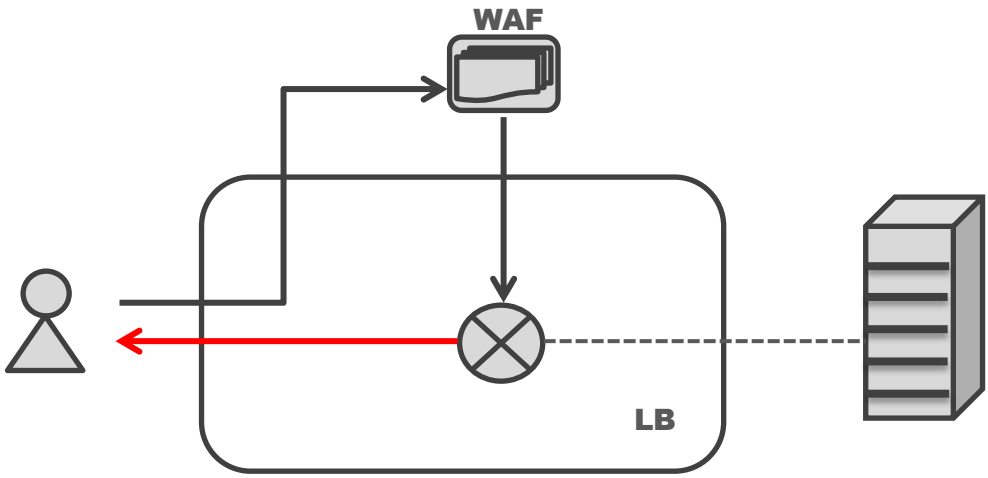

Figura 1. Balanceador de carga bloqueando petición HTTP.

Para ello el desarrollo se apoyará en software libre, que consistirá en un software de balanceo llamado Pound, con licencia GPL (General Public License), y una librería de análisis de tramas HTTP llamada Libmodsecurity, con Licencia Apache. Ya que la librería necesita obtener los datos HTTP de forma ordenada, se aprovechará el parseo HTTP que hace Pound durante el procesamiento de las peticiones.

Como herramienta para la configuración de esta solución, se creará una API HTTP en Perl, que permita de una format intuitiva gestionar las reglas WAF. Esta API podría ser consumida posteriormente por un recurso Web, que dotará al sistema de una interfaz gráfica de gestión.

Por último se validará el desarrollo mediante:

- Una herramienta de tests funcionales para la API.
- Casos prácticos de detección y prevención de algunos ataques Web.
- Tests de rendimiento de la integración.

2 ESTADO DEL ARTE

El proyecto gira entorno a dos ejes principales, los balanceadores de carga y los sistemas WAF (Web Application Firewall), por ello en este punto se detalla en qué consiste cada una de estas soluciones y se indagará en algunas de las implementaciones que ahora mismo hay disponibles en el mercado.

2.1 Balanceador de carga

Un balanceador de carga es un dispositivo de red. Su función principal es la de hacer de elemento virtualizador del servicio, una interfaz a la que se conectan los usuarios, para posteriormente ser repartidos entre distintas unidades reales del servicio.

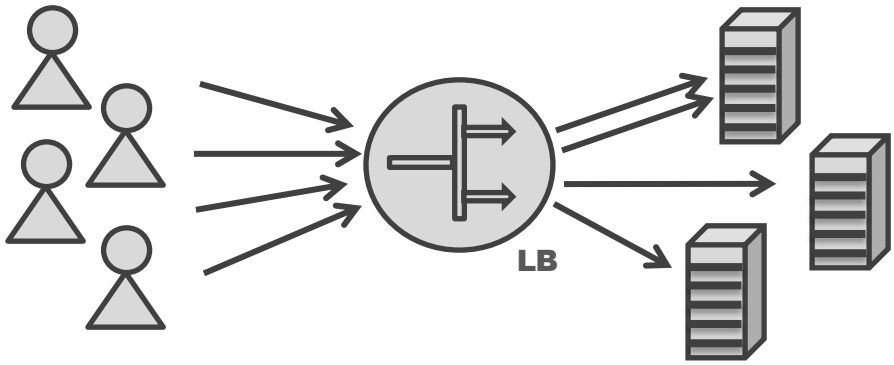

Figura 2. Balanceador repartiendo tráfico entre un pool de backends.

Otras de las funciones importantes del balanceador, es la de comprobar el estado de todos los recursos que tiene disponible, así si alguno de ellos estuviese saturado, o fuera de servicio, el balanceador no reenviaría más peticiones a este recurso.

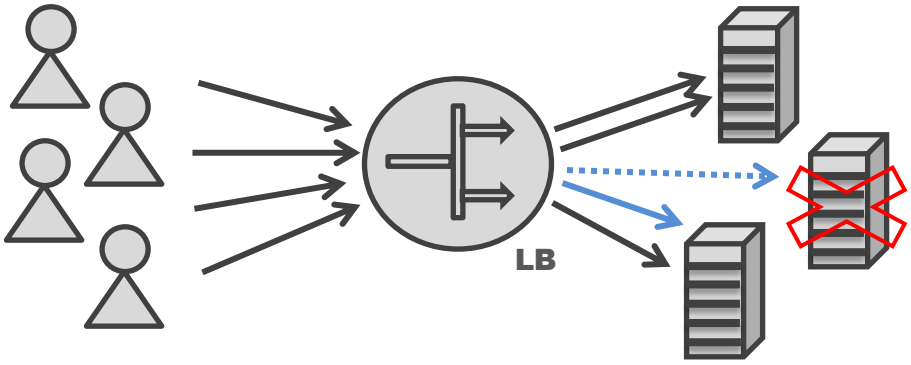

Figura 3. Balanceador redirigiendo tráfico tras fallo en un backend.

La última de las propiedades que hacen al balanceador tan popular, es la rápida escalabilidad. La topología puede cambiar, deducirse o aumentar, en cuanto a procesamiento, con un bajo coste operativo y manteniendo el servicio sin cortes.

2.1.1 Balanceadores L4 y L7

A la hora de hablar sobre balanceadores, encontramos que existen dos tipos: L4 (Layer 4) y L7 (Layer 7). Con la palabra Layer nos hace referencia a la capa del modelo OSI en el que trabajan.

Así el balanceador L4 trabaja en la capa de transporte, éstos actúan como un NAT, transformando la IP y puerto **virtual** del balanceador, en la IP y puerto **real** del recurso al que será redirigido. Al trabajar en una capa inferior a los L7, la complejidad del balanceo será menor, ya que aún no hay acceso a datos de nivel de aplicación.

El balanceador L7, por el contrario, desempaqueta los datos de aplicación, por lo que tiene acceso a las cabeceras HTTP. Al haberlas desempaquetado, puede modificarlas, o incluso añadir o quitar. Su uso más común es como proxy HTTP pudiendo multiplexar varias aplicaciones en un solo servicio y redirigir la petición por su URI. A su vez balancean entre una lista de servidores que pueden alojar la petición.

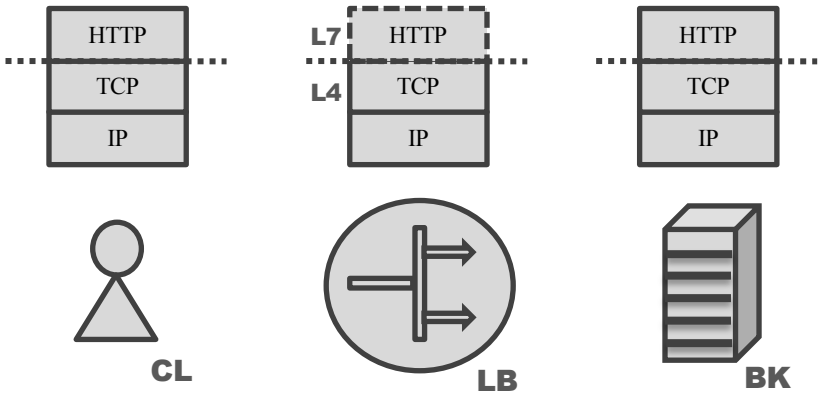

Figura 4. Modelo de capas de un balanceador de carga

2.1.2 Comparativa de soluciones para balanceo de carga

Existen diferentes implementaciones para balanceo de carga dependiendo de las necesidades de cada usuario. Algunas de ellas están enfocadas en mover paquetes basándose en los datos obtenidos de las conexiones. Éstas corresponderían a balanceadores de carga en capa 4, dan soporte a múltiples propiedades pero sin poder profundizar en soluciones complejas. Por ejemplo es posible establecer persistencia por IP pero no por *cookie* o sesión.

LVS (Linux Virtual Server) es un proyecto que está integrado en el kernel de Linux. Es capaz de repartir conexiones desde el mismo kernel, sin tener que usar datos de espacio de usuario que consumen mayor cantidad de recursos. Es un proyecto maduro que lleva desarrollándose desde 1998.

Nftlb (Netfilter load balancer) es un proyecto basado en la nueva generación de firewalls de Linux, NFtables.

Utiliza propiedades del kernel de Linux y marcado de paquetes para enrutar las conexiones de clientes hacia el pool de servidores. Los *benchmarks* lo posicionan como una de las soluciones más óptimas disponibles. Dispone de un socket de control que permite modificar las reglas de balanceo en caliente, sin cortar el servicio, a través de objetos JSON.

Pen es una solución de balanceo que utiliza datos de conexión para mover conexiones en espacio de usuario. Esto le otorga un rendimiento más modesto que las soluciones implementadas en el kernel.

Las soluciones descritas hasta el momento son más óptimas pero no tienen acceso a los datos de aplicación. Estos datos son los que pretende inspeccionar el WAF (Web Application Firewall) para analizar si se está produciendo una amenaza. Para ello es necesario utilizar una solución que desempaquete los datos de aplicación como son las que trabajan en espacio de usuario con tramas HTTP.

Nginx es uno de los servidores Web más usados para publicar contenido en Internet. Es un software robusto que ofrece mejores prestaciones que su competidor directo Apache. Nginx, puede trabajar como proxy inverso, derivando las peticiones HTTP en otros servidores, incluso aplicando algoritmos de balanceo de carga. Aun así sigue siendo un servidor Web, lo que quiere decir que implementa lógica de servidor Web aunque no sea usada. Esto añade complejidad al desarrollo ya que el código crece considerablemente.

HAproxy es una de las soluciones más utilizada como proxy inverso y balanceador de carga. Dispone de un gran rendimiento y ofrece multitud de posibilidades de configuración. Aunque debido a la envergadura del proyecto puede resultar complejo integrarse con él.

Pound es un proyecto con características exclusivas de proxy inverso y balanceo de carga. Es un proyecto simple y mantenido que dispone de un buen rendimiento, llegando a ser parejo al de Nginx cuando es usado como proxy inverso.

Todos los proyectos descritos son de código libre, teniéndose disponible su código en el repositorio correspondiente de cada uno de los proyectos.

2.1.3 Pound

Pound será la solución de proxy inverso elegida como balanceador de carga. A diferencia de otros balanceadores del mercado que puedan tener las mismas capacidades, Pound se presenta como un software ligero, que no implementa lógica de servidor de contenidos, por lo que lo hace más ligero y fácil de tratar a nivel de código.

Presenta las siguientes características:

1. Proxy inverso. Pasa peticiones desde el cliente hasta el servidor de aplicación.
2. Balanceador de carga. A la hora de elegir el servidor de aplicación, puede elegir entre una lista. Mantiene las sesiones entre cliente y backends.
3. Desencripta SSL. Puede desencriptar tráfico HTTPS y mandárselo en claro, HTTP, a los backends.
4. Comprueba que las tramas HTTP/HTTPS estén bien formadas.
5. Comprueba si los backends están operativos, si no es así no les manda peticiones.
6. Puede redirigir peticiones a determinados backends en función de la URL.

Pound está escrito en C y optimizado para Linux. A partir de su versión 1.0, Pound es apto para entornos de producción.

2.2 Web Application Firewall

Un WAF, al igual que un firewall convencional trabaja buscando patrones, y aplicando acciones en función de unas reglas previamente definidas por el administrador de seguridad. La diferencia se encuentra en que el WAF no busca patrones en el tráfico de paquetes o las conexiones, los Web Application Firewall, analizan las transacciones HTTP, e interpretan datos que van dirigidos hacia aplicaciones con el fin de detectar las amenazas.

2.2.1 Comparativas de soluciones para Web Application Firewall

Dentro del catálogo de soluciones disponibles encontramos unas de código abierto y otras de código privativo. Las soluciones privativas no dan la posibilidad de acceso al código por lo que se hace imposible su utilización para ser integradas con el balanceador de carga. Aun así, algunas de las más conocidas son:

- **Incapsula**. Es una solución compacta, formada por un balanceador de carga y una capa de protección WAF. Es vendida como servicio para aplicaciones *Cloud* o como software para su uso como dispositivo de red.
- **Kemp**. Es una empresa conocida en el mercado por sus productos de balanceadores de carga. Kemp también dispone de un módulo WAF (Web Appication Firewall), el cual está basado en el proyecto de software libre Modsecurity.

Las soluciones privativas vistas, son proyectos completos que encapsulan por un lado un balanceador de carga y a su vez, una capa que permita la inspección de contenido. Como soluciones *open source* destacan las siguientes:

- **Modsecurity**. Es un proyecto que nació como módulo de Apache y que es el encargado de dotar de seguridad a las aplicaciones Web que éste sirve, comprobando reglas de seguridad con el tráfico que la aplicación recibe. Estas reglas siguen una estricta sintaxis llamada SecLang. El módulo, debido a que estaba siendo exportado a otros servidores Web como Nginx o ISS (Internet Information Services) dio un cambio a inicios de 2018, pasando de ser un módulo de Apache, a una librería que pudiese ser integrada en cualquier servidor o proxy HTTP y que dejaría de tener las pesadas dependencias de Apache. A esta librería se le llama **Libmodsecurity**.
- **Ironbee**. Es una librería que surgió como intento de estandarizar otros motores WAF que ya existían. Parte de su sintaxis de reglas está basada en la de Libmodsecurity, SecLang. Actualmente lleva desmantenida desde 2016.

2.2.2 Libmodsecurity

La solución elegida para implementar el WAF es Libmodsecurity. Libmodsecurity es una librería escrita en C/C++. Es un software que lleva en el mercado desde 2006 y que es usado por muchos fabricantes. Nació debido a las limitaciones que presentaba SNORT para el análisis de contenido HTTP.

La gran potencia de Libmodsecurity es su lenguaje de reglas, llamado SecLang. Mediante reglas que siguen este formato se pueden aplicar infinidades de patrones de búsquedas en los mensajes, a la vez que permite

aplicar multitud de acciones.

El lenguaje SecLang funciona mediante directivas. Las directivas son las sentencias que definen acciones o configuraciones para la librería. Algunos ejemplos son: *SecAction*, *SecDebugLog* o *SecMarker*. La más utilizada es *SecRules*, que aplica una acción en consecuencia de que se cumpla un previo patrón. El formato de las directivas es una cadena con la directiva y su valor, a excepción de *SecRules* que se componen de la siguiente forma: SecRule "Variables" "Operador" "Acciones". La directiva debe estar definida en una única línea, aunque puede utilizarse el carácter '\' para unir dos líneas. Unos ejemplos serían:

```
SecMarker "END-RESPONSE-953-DATA-LEAKAGES-PHP"
SecRule &TX:crs_exclusions_wordpress|TX:crs_exclusions_wordpress \
    "@eq 0" \
    "id:9002000,\
    phase:1,\
    pass,\
    t:none,\
    nolog,\
    skipAfter:END-WORDPRESS"
```

Para formar una regla, *SecRule*, se necesitan 3 parámetros:

1- **Operator**. Aquí va definido lo que queremos buscar en la variable y como lo queremos hacer, por ejemplo, "*@rx (server|servidor)*", para buscar la palabra *server* o *servidor* en una **Variable**. Es donde se va a realizar la búsqueda. La librería las va rellenando conforme va recibiendo los trozos de las tramas. A parte de las que define la librería, ésta permite crear las nuestras propias. Hay 105 variables diferentes que se dividen en las categorías: *request*, *response*, *server*, *time*, *collections* y *miscellaneous.*

2- **Variable**. Existen 36 operadores disponibles que se dividen en: *string*, *numerical*, *validation* y *miscellaneous.*

3- **Actions**. Se aplicará en el momento que el operador sea encontrado en la variable. Existen 47 acciones y se dividen en los siguientes tipos: *disruptive*(son las más importantes, ya que son las que pueden detener en tráfico HTTP), *flow*, *metadata*, *variable*, *logging* y *miscellaneous.*

Libmodsecurity tiene 5 fases de funcionamiento: *Request Header*, *Request Body*, *Response Header*, *Response Body* y *Logging*. Las reglas se aplicarán en una de ellas. Con lo que el administrador de seguridad, debe ser responsable de elegir correctamente la fase en función del contenido que desee analizar, de lo contrario puede que las variables respectivas aún no hayan sido rellenas.

Los mayores inconvenientes de la librería vienen del lenguaje SecLang, ya por su complejidad debido a la gran cantidad de opciones posibles, o a que una mala gestión de éstas puede hacer que se consuman muchos recursos. Ante esto, existen empresas especializadas en vender paquetes de estas reglas, al igual que algunos de acceso libre como los suministrados por la organización OWASP (Open Web Application Security Project).

3 DISEÑO DE LA IMPLEMENTACIÓN

Durante el diseño del desarrollo se distinguirán dos partes diferenciadas. La primera de ellas consiste en dar soporte al balanceador de WAF (Web Application Firewall), para esta etapa se utilizará el lenguaje C y consistirá de dar todas las características que permite Libmodsecurity a Pound.

La segunda etapa es el diseño e implementación de la API de configuración. Esta API será montada sobre un servidor Web y será la encargada de recibir peticiones HTTP del operador de seguridad y transformarla en sentencias SecLang que la librería pueda interpretar y cargar para el análisis del tráfico.

3.1 Integración entre balanceador y Web Application Firewall

La primera de ellas, es la integración entre Pound y Libmodsecurity. El primer paso a alcanzar será el de cargar los ficheros de reglas en estructuras de datos que Pound entienda. Se hará de forma que las reglas puedan ser recargadas en un futuro mediante una opción de *reload*, ya que por el carácter crítico del balanceador de carga, no debe detenerse el servicio.

Otra de las opciones que se añadirá, será que Pound pueda comprobar la sintaxis de una regla, o un fichero de reglas, SecLang. Con esto se pretende dar una herramienta con la que comprobar que las reglas son correctas antes de añadirlas al fichero de configuración.

Respecto al tema central, el paso de datos de Pound a Libmodsecurity, cada vez que Pound reciba los datos completos de una fase, los parseará y se los mandará a Libmodsecurity. Libmodsecurity esperará hasta que se le de la orden de que los datos de la fase están completos y puede comenzar a analizar el tráfico. Cuando acabe, la librería responderá a través de una estructura, llamada interruption, aquí es donde comunica su resolución.

Los campos que componen esta estructura, y por tanto las acciones que debe acatar Pound son:

- **Disruptive**. Alguna regla ha dictado que la transacción debe terminar.
- **Url**. La resolución es un redirect, la comunicación termina y hay que enviarle un mensaje HTTP *redirect* al cliente
- **Code**. Es el código de estado HTTP que hay que enviarle al cliente, tanto para el *redirect* como para denegarle la petición.
- **Log**. Es el mensaje de debe escribirse en el archivo de logs del balanceador.

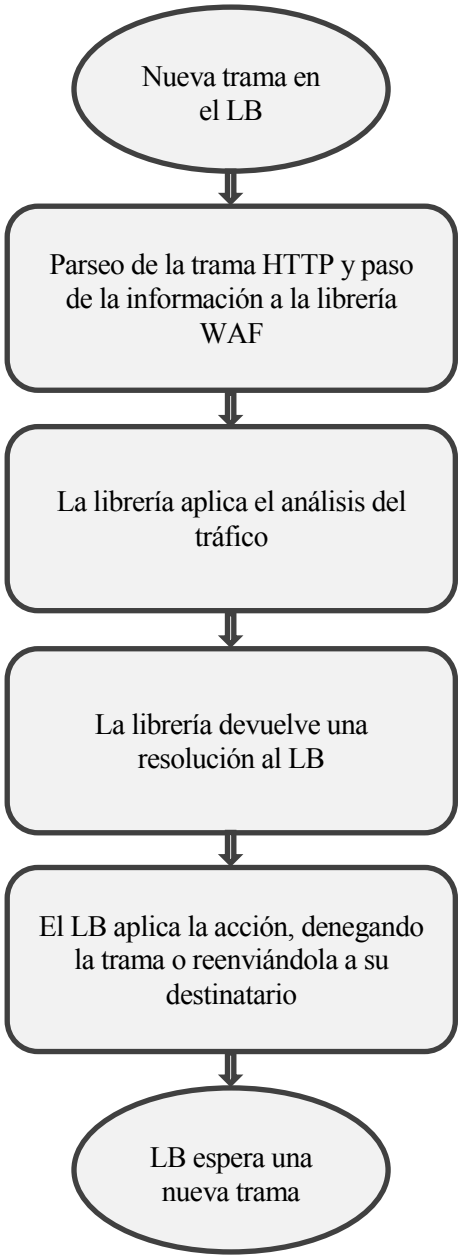

Figura 5. Lógica del balanceador y el WAF

3.2 API de configuración

Una vez completa la primera parte del desarrollo, se pasará a crear una API de gestión del sistema, que permita gestionar las reglas y recargue los ficheros de forma autónoma. Esta segunda parte consistirá en un desarrollo con metodología modelo-controlador. El modelo validará los datos de entrada, mientras que el controlador, hará la lógica de más bajo nivel, modificando los ficheros de configuración.

Esta API estará escrita en Perl, con una metodología REST, y será gestionada por un *script* servido por un servidor Web. Las llamadas a la API serán llamadas HTTP.

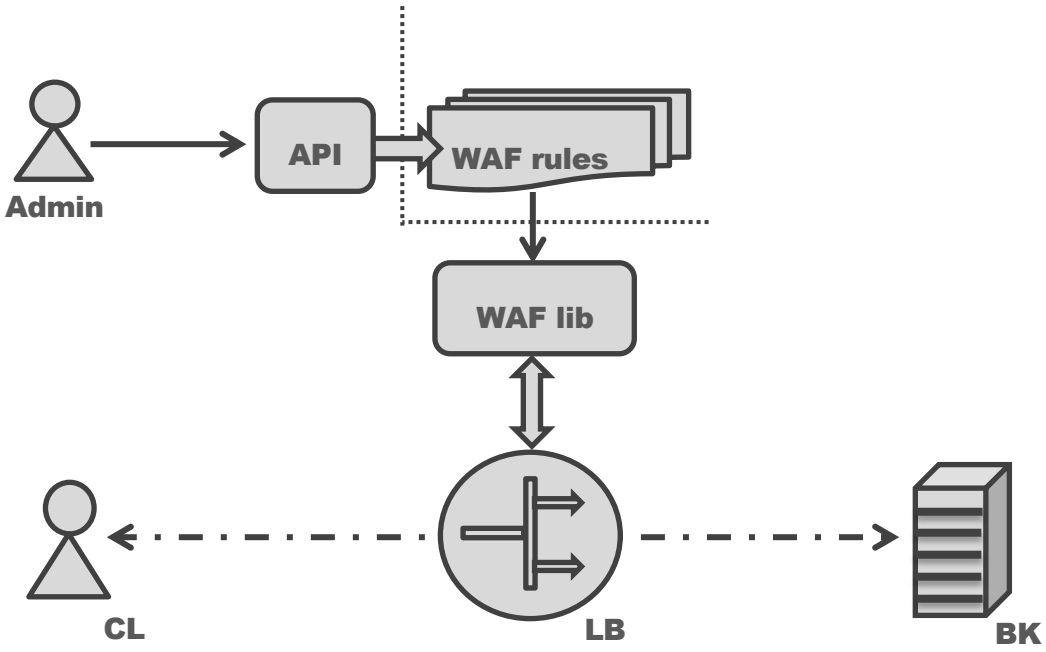

Figura 6. Interacción entre la API y el WAF

4 INTEGRACIÓN ENTRE BALANCEADOR Y WAF

Para realizar la integración, el primer paso será conocer el comportamiento de libmodsecurity y adaptarlo a la arquitectura de Pound.

Libmodsecurity está formado por cuatro objetos principales: *modsec*, *rules*, *transaction* e *interruption*.

- *Modsec*, es el elemento que crea e inicializa la estructura con la que comunicarse con la API de Libmodsecurity.

- *Rules*, este objeto es el encargado de almacenar todos los *sets* de reglas que han sido cargados.

- *Transaction*, es quién gestiona el paso de datos HTTP en una petición-respuesta. Las transacciones guardan las variables usadas por las reglas, las crean y les asignan valores en función de los paquetes HTTP recibidos. Cuando una petición-respuesta entre el cliente y el LB (Load Balancer) acabe, la transacción debe ser eliminada para que se liberen todos los recursos asociados a la petición y la respuesta. Por cada una de las fases de una transacción es necsario aplicar los siguientes pasos:

    - Indicar a la librería dónde se encuentra la información que ésta necesita, referenciándole cada uno de los datos HTTP recibidios, a través de funciones como: *msc_add_n_request_header*, *waf_add_http_info* o *msc_append_request_body*.

    - Dar la orden de procesar a Libmodsecurity, una vez que conoce los datos que debe analizar. Se utilizan las funciones: *msc_process_uri*, *msc_process_request_body*, *msc_process_request_headers*...

- *Interruption*, es el objeto donde se guarda la resolución de una transacción y la acción que debería ser aplicada. Mediante la función *msc_intervention* es posible conocer el estado actual de una transacción.

En el esquema de Pound se distinguen dos etapas de procesamiento. La primera es la inicialización de los módulos de Pound, en función de la configuración con la que se arrancó el binario. Una vez todo está configurado el proceso entra en un bucle el cual comprueba si ha llegado alguna petición nueva. En caso de que así sea, la petición es asignada a un hilo de procesamiento y encolado hasta que pueda ser gestionado. La segunda etapa es el la gestión del hilo que maneja el paso de mensajes entre el cliente y el servidor, añadiendo la lógica de balanceo y donde se añadirán comprobaciones del WAF (Web Application Firewall).

Para aplicar acciones a un proceso de Pound que se encuentre corriendo, Pound cuenta con una herramienta llamada Poundctll encargada de comunicarse mediante un socket de control y enviarle comandos. Algunas de las acciones que Poundctl permite son: obtener estadísticas de conexiones, añadir backends o poner backends en mantenimiento.

El esquema correspondiente a la primera etapa de procesamiento de Pound puede verse en la siguiente figura:

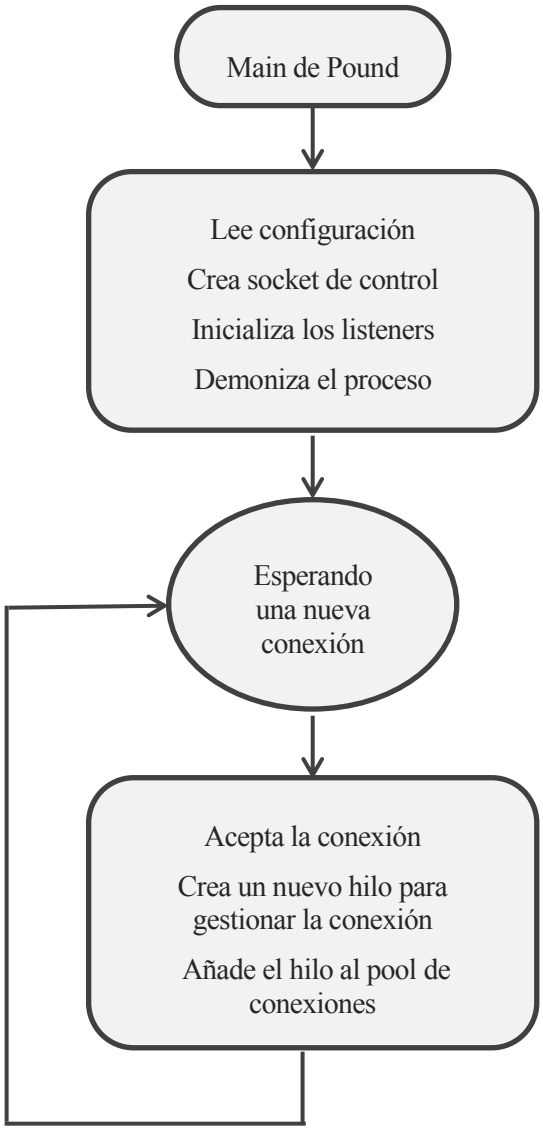

Figura 7. Diagrama del main de Pound

Las conexiones de Pound son gestionadas por un esquema de "una conexión, un hilo". Este esquema reserva en la memoria del programa un espacio en el que se añadirán todos los datos correspondientes a la conexión. El hilo dependiendo del tipo de petición que haya llegado del cliente podrá tener un tiempo de vida mayor o menor. Por ejemplo, en el caso de HTTP/1.0 el hilo muere una vez el balanceador a reenviado la respuesta del servidor al cliente, mientras que en el caso de una petición corriente de HTTP/1.1 el canal podrá quedarse abierto hasta que o bien lo cierre el cliente o el *backend*.

Esta lógica de "una conexión, un hilo" presenta un problema ya conocido, y es la penalización por interbloqueos, ya que cuando el programa cambia de contexto, de un hilo a otro, se producen bloqueos de acceso a la memoria. Cómo ventaja, presenta un esquema más sencillo que otros, cómo podría ser la programación por eventos.

A continuación puede verse el diagrama de flujo simplificado que sigue Pound en la gestión de una conexión HTTP. En esta memoria no se tratará la lógica de balanceo o de comprobación del estado de los *backends*, ya que es transparente para la inspección de tráfico.

(CL <- LB) Abre el socket SSL con el cliente.
(CL -> LB) Obtiene las cabeceras del cliente.
Calcula servicio y backend al que debe dirigirse la petición.
(LB -> BK) Conecta con el backend elegido.
(LB -> BK) Envía las cabeceras al backend.
(LB -> BK) Añade cabeceras de proxy.
(CL -> LB -> BK) Reenvía el body de cliente a backend.
(LB <- BK) Obtiene las cabeceras del backend.
(CL <- LB) Envia las cabeceras al cliente.
(CL <- LB) Añade cabeceras proxy al cliente.
(CL <- LB <- BK) Reenvía el body de backend a cliente.
Log del resultado de la comunicación.
Si el protocolo es HTTP/1.1, se queda eperando una nueva petición.

Figura 8. Secuencia de acciones de do_http

El código de Pound se encuentra repartido en los siguientes ficheros:

- *config.c*, son las funciones que configuran Pound, trata tanto los argumentos de entrada, como el fichero de configuración.
- *http.c*, tiene la lógica para la gestión de una la petición y respuesta entre un cliente y un *backend*.
- *svc.c* y *svc.h*, son funciones donde se implementa la lógica de balanceo de carga y de elección del *backend* que servirá la petición del cliente. Implementa algunas características como: el control de sesiones, manipulación de cookies o funciones usadas en el proceso de Poundctl.
- *pound.h*, son las definiciones de las estructuras de datos.
- *pound.c*, main del proceso Pound.
- *poundctl.c*, main del Proceso Poundctl.
- *waf.c*, son las nuevas funciones implementadas para las características de WAF.
- *pound.8* y *poundctl.8*, manuales de los binarios Pound y Poundctl respectivamente.
- *config.h.in, configure.in*, *Makefile.in*, ficheros de configuración de la herramienta autotools.
- *dh512.h, dh1024.h*, definiciones para crear certificados de 512 y 1024 bytes respectivamente.

En cuanto a la nomenclatura utilizada para el desarrollo de las nuevas funcionalidades de Pound, se ha seguido la siguiente estandarización.

- Las nuevas funciones y variables comienzan por la etiqueta "**waf_**". Estas funciones suelen apoyarse en funciones de Libmodsecurity por lo general. Durante el desarrollo del punto 2 de la memoria, se irán dando puntualizaciones sobre cómo y dónde son usadas. Para una explicación más detallada, de cómo actúa cada una de estas funciones, ver el Anexo 1.
- Las funciones de Libmodsecurity tienen su propio nombrado, usándose la etiqueta "**msc_**". Las funciones que han sido usadas de Libmodsecurity se explican en el Anexo 1.

4.1 Makefile y compilación condicional

A continuaciónn se adecuará el código de Pound para que pueda enlazar con la nueva librería, Libmodsecurity. La *flag* –I indica al compilador dónde buscar los ficheros de cabeceras y la *flag* –L dónde se encuentran los ficheros compilados de la librería.

Enlace de las librerías

Fichero: *Makefile.in*

```
INCLUDES=-I/usr/local/modsecurity/include/ \
 -L/usr/local/modsecurity/lib -Wl,-rpath=/usr/local/modsecurity/lib

CC=@PTHREAD_CC@ -g -rdynamic $(INCLUDES)

LIBS=@LIBS@ @PTHREAD_LIBS@ -lmodsecurity
```

El nuevo desarrollo será condicional, para ello se utilizará una variable de compilación, que habilitará el código de WAF en caso de que la variable tenga un valor positivo durante el proceso de compilación del programa. Así, todos los bloques de código presentados en este punto 2 van englobados dentro de una sentencia *IFDEF*. Por limpieza de la memoria y legibilidad del código, estos *flags* no aparecerán en los bloques de código.

Flag de compilación

Fichero: *Makefile.in*

```
C_WAF=-DWAF=1
CFLAGS=$(CFLAGS) ${C_WAF}
OBJS=pound.o http.o config.o svc.o pound_sync.o waf.o
```

4.2 Inicialización y configuración

Pound funciona cargando parte de su configuración en variables que son globales para todo el programa. Son estructuras a las que pueden acceder otros hilos de ejecución para comprobar la configuración del sistema y así gestionar las conexiones en consecuencia. Por tanto se crearán nuevas variables con la que gestionar la configuración del WAF. Estas variables son:

- **waf_rules**, es una estructura cargada en memoria con los conjuntos de reglas configurados y parseados.
- **waf_api**, es el objeto sobre el que se crean transacciones y que gestiona la lógica de Libmodsecurity.
- **waf_rules_file**, es una lista enlazada con los *path* a los ficheros de reglas que han sido leídos del fichero de configuración de Pound.

Variables globales

Fichero: *Pound.c*

```
Rules   *waf_rules;
ModSecurity   *waf_api;
FILE_LIST *waf_rules_file;
```

Siguiendo el diagrama de flujo de Pound en el proceso de arranque, se inicializaran los nuevos objetos durante el proceso de configuración de Pound. Se obligará a que los ficheros de reglas sean correctos, si ésto no se cumpliese, el programa no continuará con el arranque.

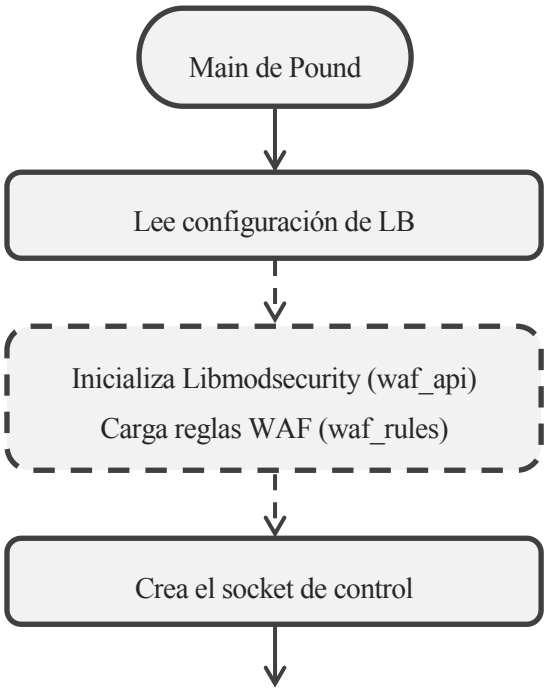

Figura 9. Insertar lógica de WAF en proceso Pound

Inicialización del módulo de WAF

Fichero: *pound.c*, función: *main()*

```
// WAF initializate waf API
waf_api = msc_init();

// load rules
if (waf_reload_rules())
  exit(1);
```

Para indicar a Pound la ruta a los ficheros de reglas de WAF, se ha añadido una nueva directiva en el fichero de configuración de Pound llamada *WafRules*. Se permite añadir tantas directivas *WafRules* como se necesiten, respetándose durante la ejecución el orden en que aparecen en el fichero de configuración. Esta directiva es parseada y guardada en la estructura de datos definida con anterioridad, llamada *waf_rules_files*.

Para parsear las directivas *WafRules* se utilizará la función *config_parse_waf*. La lógica para obtener este parámetro no se ha integrado en la función general de Pound encargada de parsear todo el fichero de configuración. Con esto se pretende poder recargar la configurción de WAF sin afectar al proceso de Pound. Ya que Pound no tiene funcionalidad de *reload*, se ha creado una exclusiva para recargar las directivas *WafRules*.

Leer la configuración del WAF del fichero de configuración

Fichero: *config.c*, función: *config_parse_waf()*

```
void
config_parse_waf() {

  char        lin[MAXBUF];
  FILE_LIST *file = NULL;
  FILE_LIST *bef_file = NULL;
  FILE_LIST *it = waf_rules_file;

  if( waf_rules_file != NULL ) {
    // Free all files struct
      while ( it ) {
          it = it->next;
          free ( waf_rules_file );
          waf_rules_file = it;
      }
      waf_rules_file = NULL;
  }
  bef_file = NULL;

  regcomp(&WafRules, "^[ \t]*WafRules[ \t]+\"(.+)\"[ \t]*$",
    REG_ICASE | REG_NEWLINE | REG_EXTENDED);

  conf_init(conf_name);

  while(conf_fgets(lin, MAXBUF)) {
    if(!regexec(&WafRules, lin, 4, matches, 0)) {
      if ((file = (FILE_LIST *)malloc(sizeof (FILE_LIST)) ) == NULL )
        conf_err("WAF config: out of memory - aborted");

      else {
        lin[matches[1].rm_eo] = '\0';
        file->file = strdup(lin + matches[1].rm_so);
        file->next = NULL;

        if(waf_rules_file != NULL)  {
          bef_file = file;
          waf_rules_file = file;
        }
        else
          bef_file->next = file;
      }
    }
  }

  regfree(&WafRules);
}
```

La opción de recargar las reglas WAF se hace mediante el binario Poundctl, usando la opción –R y el *path* al socket de control. Este *path* se define en la directiva *Control* del fichero de configuración de Pound. Un ejemplo de la ejecución podría ser: "*poundctl –R –c /tmp/ctl_socket*".

Ejecutar un *reload* de WAF no afectará a las conexiones que se encuentren establecidas, ya que la transacción encargada del análisis HTTP mantiene una copia del conjunto de reglas que existía en el momento que se creó.

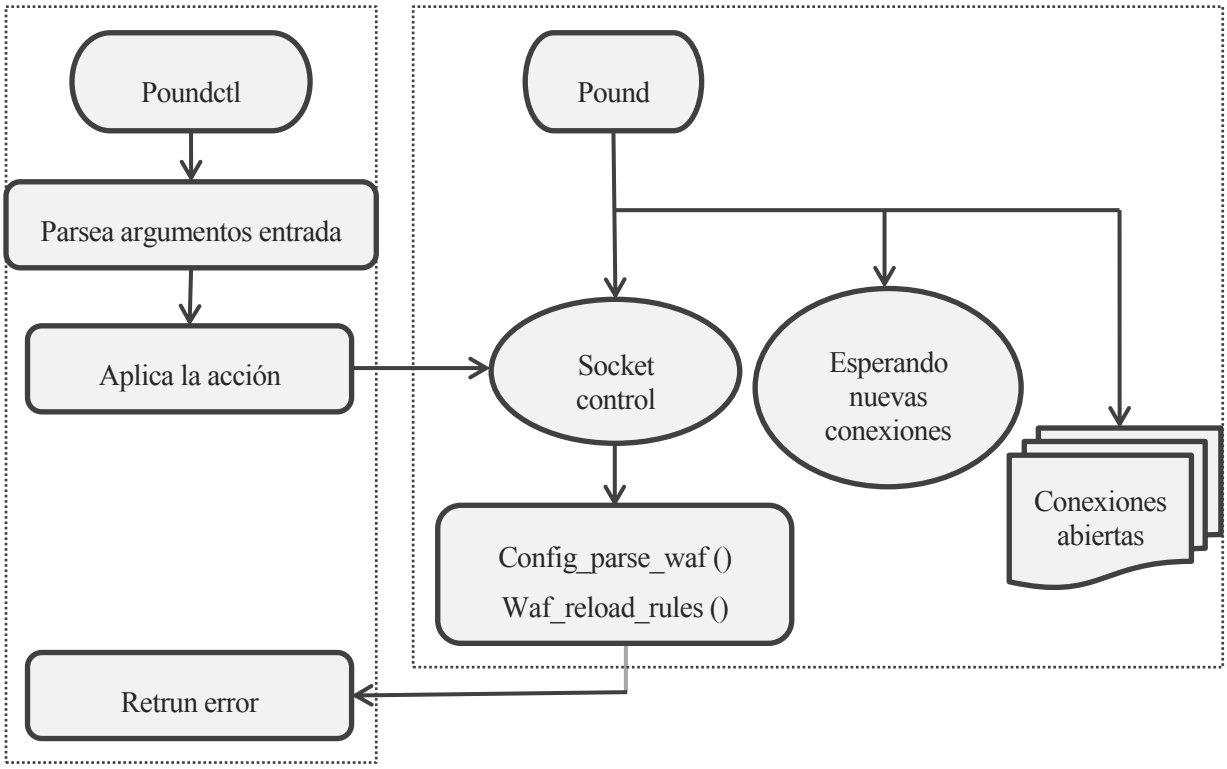

Figura 10. Comunicación entre Pound y Pounctl

Acciones admitidas a través del controlador

Fichero: *Pound.h*

```
typedef enum {
  CTRL_LST,
  CTRL_EN_LSTN,
  CTRL_DE_LSTN,
  CTRL_EN_SVC,
  CTRL_DE_SVC,
  CTRL_EN_BE,
  CTRL_DE_BE,
  CTRL_ADD_SESS,
  CTRL_DEL_SESS,
  CTRL_FLUSH_SESS,
  REL_WAF
} CTRL_CODE;
```

Al igual que muchos otros programas Linux, Pound también utiliza la función *getopt* para leer los argumentos

de entrada. La notación esperada por esta función consiste en un string en el que se encuentran todos los posibles valores de entrada. Si algún parámetro va seguido del carácter "**:**" quiere decir que ese parámetro requiere un valor, el cual irá a continuación.

Leyendo los parámetros de entrada

Fichero: *Poundctl.c*, función: *main()*

```
    while(!i && (c_opt = getopt(argc, argv, "c:LlSsBbNnXHR")) > 0)
        switch(c_opt) {
        case 'c':
            sock_name = optarg;
            break;
        case 'X':
            xml_out = 1;
            break;
#if WAF
          case 'R':
              reload_waf = 1;
              break;
#endif
        case 'L':
            if(is_set)
                usage(arg0);
            en_lst = is_set = 1;
            break;
```

Al recibir el comando en Pound

Fichero: *srv.c*, función: *thr_control()*

```
#if WAF
        case REL_WAF:
            config_parse_waf();
            waf_reload_rules();
            break;
#endif
```

La función *waf_reload_rules* es la encargada de recargar los conjuntos de reglas usados por el proceso de Pound. *waf_reload_rules* carga los nuevos conjuntos de forma segura, en el caso de que se produjese algún error, ya sea por falta de espacio o por que las reglas estuviesen mal construidas, los conjuntos volverían al valor anterior.

Debido a la complejidad de la sintaxis de SecLang, se ha desarrollado una opción para Pound, con la que arranque sólo para comprobar si las reglas o conjuntos están correctamente construidos. Para ello, Pound arrancará e intentará cargar la regla que se le manda como argumento con la opción –w, o el *path* al conjunto de reglas con la opción –W. En el caso de que consiga cargar la configuración de forma correcta, el programa saldrá con un 0. Por el contrario si detecta algún fallo, saldrá con un 1 mostrando un mensaje acorde al error. En ninguno de los casos el proxy llega a arrancar, sólo ejecuta la fase de configuración.

Al igual que en el caso de Poundctl y su opción de *reload*, Pound también usa la función *getopt* para obtener los argumentos de entrada.

Leer parámetros entrada de Pound

Fichero: *config.c*, función: *config_parse()*

```
while((c_opt = getopt(argc, argv, "sf:cvVp:w:W:")) > 0)
    switch(c_opt) {
      case 'W':
          waf_set_str = optarg;
          waf_checks = 2;
          break;
      case 'w':
          waf_rule_str = optarg;
          waf_checks = 1;
          break;
```

```
.....
```

```
// Check rule and rule_set
if (waf_checks) {
  if (waf_checks == 1)
      waf_err = waf_check_rule(waf_rule_str);
  else if (waf_checks == 2)
      waf_err = waf_check_set(waf_set_str);
    exit(waf_err);
}
```

4.3 Transacción

La transacción es el objeto que utiliza Libmodsecurity para gestionar el par de mensajes HTTP que consisten en una petición y su respuesta. Para analizar una transacción por completo, es necesario procesar los siguientes datos:

En la petición del cliente:

- La conexión, se refiere a la IP y el puerto tanto del cliente como del LB.
- La URI, identifica el objeto o recurso al que se quiere acceder dentro del servicio.
- El método HTTP, es la acción que se quiere realizar sobre el recurso. También llamado verbo.
- La versión, es la versión del protocolo HTTP.
- Las cabeceras de la petición, son las cabeceras enviadas por el cliente.
- El cuerpo de la petición, son los datos de nivel de aplicación enviados por el cliente.

En la respuesta del backend:

- El código de estado, es el código HTTP de respuesta del servidor, usado para identificar el resultado

obtenido de haber procesado la petición.

- Las cabeceras de la respuesta, son las cabeceras enviadas por el servidor.

- El cuerpo de la respuesta, son los datos de aplicación que devuelve el servidor.

Debido al procesamiento que sigue pound para cada cliente, un hilo por cada conexión, deberá existir una transacción en cada uno de estos hilos. La transacción debe morir en el momento que es procesada la respuesta del servidor o siempre que se corte la conexión, ya sea por una desconexión del cliente o servidor, un *timeout* o haberse dictado un fin de conexión por parte del WAF.

En el caso en que el protocolo HTTP esté trabajando con la versión 1.1, el hilo de conexión entre el cliente y el backend se queda abierto esperando una nueva petición. Para este caso, hay que asegurar que se crea una nueva transacción cuando se reciba una nueva petición, y que no existen dos transacciones abiertas simultáneamente.

Para asegurar que no existen *memory leaks* y que la transacción se borra siempre que muere el thread, se añadirá la función encargada de eliminar la transacción, *waf_del_transaction,* a la macro de Pound que libera todos los recursos del hilo, *clean_all*. El puntero encargado de guardar la estructura de la transacción siempre será inicializado a *NULL,* y configurado con este valor en el momento que se libera la transacción. Así, comprobando que el valor sea distinto a *NULL* siempre será posible saber si ya existe alguna transacción creada. Esto es necesario debido a que *clean_all* puede ser llamada en cualquier momento a lo largo del procesamiento de la función *do_http*, por lo que nunca se sabe con certeza el estado de la transacción.

Ampliando la macro de limpieza

Fichero: *http.c*, macro: *clean_all*

```
#define clean_all() {   \
  if(flagCount) {decrease_backend_conn(cur_backend); } \
  if(ssl != NULL) { BIO_ssl_shutdown(cl); } \
  if(be != NULL)
    { BIO_flush(be); BIO_reset(be); BIO_free_all(be); be = NULL; } \
   if(cl != NULL)
     { BIO_flush(cl); BIO_reset(cl); BIO_free_all(cl); cl = NULL; } \
   if(x509 != NULL) { X509_free(x509); x509 = NULL; } \
   if(ssl != NULL) { ERR_clear_error(); ERR_remove_state(0); } \
   if(body_buff != NULL) {free(body_buff); body_buff = NULL; } \
   waf_del_transaction(&modsec_transaction); \
 }
```

La función encargada de gestionar todo el paso de mensajes HTTP es *do_http*. Esta función es el *main* de cada uno de los hilos que manejan una conexión entrante. En cuanto a código, *do_http* es una función densa y poco modularizada. Es posible encontrar un esquema de esta función al comienzo del punto 2 de la memoria.

4.3.1 Gestión de las cabeceras HTTP

Pound durante recepción de la petición del cliente, separa las cabeceras en un vector, utilizando la primera posición para la cadena formada por el verbo, la versión y la URI. Para el resto del vector se utiliza una cabecera por cada una de las posiciones. Para traducir esta información a un formato de datos entendido por Libmodsecurity se han utilizado las siguientes funciones:

- *waf_add_http_info*, es la encargada de separar el verbo, la versión y la URI en tres datos independientes y enviarselos a la librería.

- *parse_headers*, separa una cabecera en un par clave-valor. Para no utilizar más memoria, se utilizan punteros que hacen referencia a donde empieza la clave y donde empieza el valor dentro de la cabecera guardada por Pound. Estos punteros son los que se envían a Libmodsecurity.

Durante esta etapa se utilizan las funciones de libmodsecurity:

- *msc_process_uri*, procesa la información referente a las URI, verbo y versión.
- *msc_add_n_request_header*, añade las cabeceras a la transacción actual.
- *msc_process_request_headers*, procesa la fase 1 de Libmodsecurity.

4.3.2 Gestión del cuerpo HTTP

Una vez terminado el procesamiento de cabeceras, se pasará al procesamiento del cuerpo HTTP. En este caso, Pound no guarda nada de esta información, Pound pospone la recepción del cuerpo hasta que le ha pasado las cabeceras al *backend*. Por ello se ha implementado una lógica encargada de guardar en un buffer el cuerpo HTTP para analizarlo con la librería de WAF y poder posteriormente reenviarlo al *backend*.

Debido a todas las posibilidades que existen a la hora de recibir el cuerpo, se han tomado varias consideraciones, recogidas en la función *waf_body_enabled*, en los que el cuerpo no se analizará en los casos que:

- No exista cabecera *content-length*.
- El body venga troceado, *chunked*.
- El body sea desactivado en el fichero de configuración de Pound, directiva *WafBodySize* puesta a 0.
- El body es mayor que el máximo impuesto en el fichero de configuración de Pound, directiva *WafBodySize*.
- Se esté usando el protocolo *RPC*.

Si el cuerpo cumpliese alguno de los patrones de la lista no sería analizado, sería aceptado con el objetivo de evitar falsos positivos.

La nueva directiva *WafBodySize*, del fichero de configuración de Pound, controla el tamaño máximo en bytes que queremos almacenar para un body HTTP. Si la directiva es puesta a 0, el body queda deshabilitado y no es examinado. Si el body va a ser analizado, se guardará en un buffer dinámico, con el objetivo de reservar la memoria precisamente necesaria, ya que podría dispararse en los casos que el balanceador manejara gran concurrencia de tráfico.

Las funciones de Libmodsecurity utilizadas para tratar el cuerpo HTTP son:

- *msc_append_request_body*, para añadir el body a los datos de procesamiento de la librería.
- *msc_process_request_body*, para ejecutar el procesamiento de la fase 2 de Libmodsecurity.

Análisis del cuerpo HTTP

Fichero: *http.c*, función: *do_http()*

```
if (waf_body_enabled(body_max_size, buf_log_tag,
    cont, chunked, is_rpc)) {
  waf_body=1;
  body_size=read_body(cl, &body_buff, cont);
  msc_append_request_body(modsec_transaction,body_buff,
      body_size);
  msc_process_request_body(modsec_transaction);
```

```
    }
```

4.3.3 Gestión de la sentencia del WAF

Una vez procesada por Libmodsecurity la petición o la respuesta, es posible obtener el resultado del análisis de las tramas. Libmodsecurity define la estructura *intervenction* en la que se almacena la sentencia dictada por la librería una vez finalizado el análisis. Los campos de la estructura son:

- *Status*, guarda el código HTTP que debe ser enviado al cliente.
- *Url*, si este campo viene relleno, contendrá la URL a la que redirigir al cliente.
- *Disruptive*, si este campo tuviese un valor positivo la trama debe ser bloqueada.

En consecuencia al valor de los campos de *intervenction*, Pound debería implementar las siguientes acciones.

Acciones definidas

Fichero: *Pound.h*

```
Typedef enum{
  ALLOW,
  REDIRECTION,
  BLOCK
}WAF_ACTION;
```

Debe hacerse una integración de la estructura *intervention* con el proxy HTTP, ya que es necesario actuar sobre el cliente, ya sea reenviándole a la URL, cambiando el código de estado de la respuesta o cortando la conexión. Todas estas acciones están fuera del espacio de actuación de la librería.

Procesando la resolución del WAF

Fichero: *http.c*, función: *do_http()*

```
waf_action = waf_resolution(modsec_transaction,&waf_code, &buf);
    if (waf_action != ALLOW) {
      if (waf_action == BLOCK)
        err_reply(cl, h403, "replied forbidden");

      else if (waf_action == REDIRECTION)
        redirect_reply(cl, buf, waf_code);

      // close conn
      logmsg(LOG_INFO, "%s (%lx) replied by WAF", pthread_self());

      free_headers(headers);
      clean_all();
      return;
    }
```

4.3.4 Procesamiento de una petición HTTP

En este apartado puede verse el el código resultante de aplicar los tres puntos anteriores. Primero se analizan las cabeceras, a continuación los datos del cuerpo y por último se aplica la acción que determina el WAF. Es importante puntualizar que la transacción debe crearse en el momento que se reciben las cabeceras del cliente.

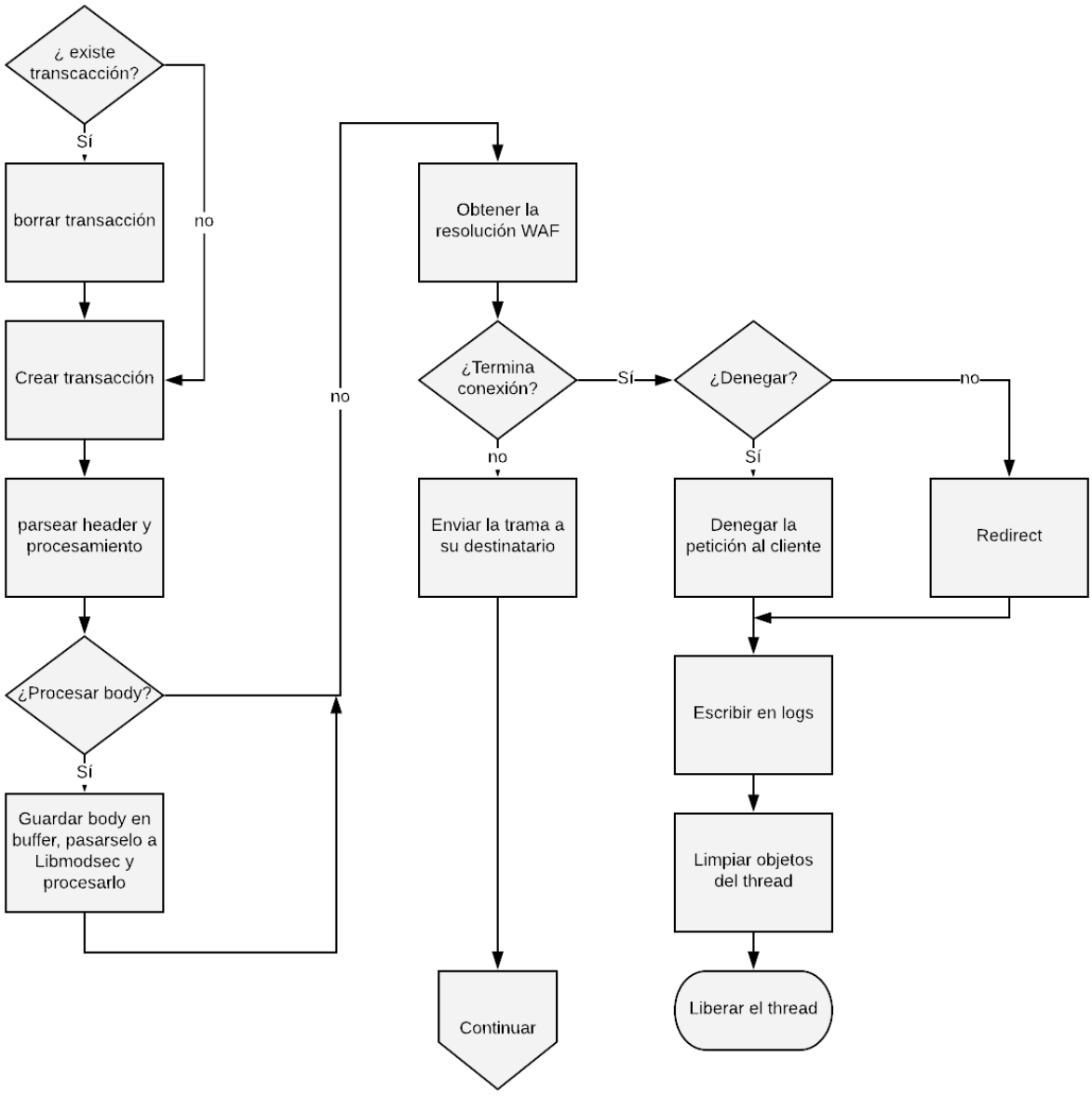

Figura 11. Esquema de una petición HTTP

Procesando la petición

Fichero: *http.c*, función: *do_http()*

```
// check WAF
```

```
#if WAF
  if (waf_rules) {
    if (modsec_transaction)
      waf_del_transaction(modsec_transaction);
    waf_create_transaction(&modsec_transaction, waf_api, waf_rules);
    waf_add_req_head(modsec_transaction, headers, headers_num);

    if (waf_body_enabled(body_max_size, buf_log_tag,
        cont, chunked, is_rpc)) {
      waf_body=1;
      body_size=read_body(cl, &body_buff, cont);
      msc_append_request_body(modsec_transaction,body_buff,
          body_size);
      msc_process_request_body(modsec_transaction);
    }

    waf_action = waf_resolution(modsec_transaction,&waf_code, &buf);
    if (waf_action != ALLOW) {
      if (waf_action == BLOCK)
        err_reply(cl, h403, "replied forbidden");

      else if (waf_action == REDIRECTION)
        redirect_reply(cl, buf, waf_code);

      // close conn
      logmsg(LOG_INFO, "%s (%lx) replied by WAF", pthread_self());

      free_headers(headers);
      clean_all();
      return;
    }
  }
#endif
```

Una vez que la trama HTTP ha sido analizada y si no se encontrara ningún patrón que hiciese detener la conexión:

- Se enviarían las cabeceras de cliente al backend.
- Se añadirían nuevas cabeceras si fuera requerido.
- Se mandaría el cuerpo al backend y se liberaría el buffer en el que estaba guardado.

Envío del body al backend

Fichero: *http.c*, función: *do_http()*

```
// if body has been read, it was saved in a buffer
if(waf_body) {
  if(BIO_write(be, body_buff, body_size) != body_size) {
    if(errno)
      logmsg(LOG_NOTICE, "%s (%lx) error write request pending: %s",
         buf_log_tag, pthread_self(),strerror(errno));
    clean_all();
    pthread_exit(NULL);
  }
```

```
    free(body_buff);
    body_buff=NULL;
    waf_body=0;

  /* had Transfer-encoding: chunked so read/write all the chunks
(HTTP/1.1 only) */
  } else if(cl_11 && chunked) {
```

4.3.5 Procesamiento de una respuesta HTTP

El procesamiento de una respuesta HTTP es igual al de la petición. Las únicas características que cambian son:

- No hay que crear una transacción, se usa la misma que se creó cuando se analizó la petición.
- Tras analizar el cuerpo respuesta es obligatorio borrar la transacción, ya que si llegase una nueva petición debería crearse una nueva transacción.
- En la respuesta no se procesa el verbo, URI o versión, en cambio se procesa el código HTTP.

4.4 Resumen

Como conclusión, al finalizar la implementación detallada a lo largo de este punto, la interfaz de Pound presenta los siguientes cambios:

Pound acepta dos nuevas directivas de configuración en sus ficheros de configuración:

- *WafRules*, es la ruta a un fichero con reglas WAF. Puede haber tantas directivas de este tipo como se necesite añadir, éstas serán ejecutadas en el orden que aparecen en el fichero de pound.
- *WafBodySize*, es el tamaño máximo que se comprobará para un body HTTP. Si el body excede este tamaño, se omitirá su comprobación.

El binario Pound cuenta con dos nuevos argumentos de entrada para su ejecución:

- W, para comprobar si un fichero de reglas es correcto y cumple los estándares de libmodsecurity.
- w, pound recibe como argumento una regla SecLang y comprueba que su sintaxis sea correcta.

El binario Poundctl, usado para interaccionar con un proceso ya corriendo de Pound cuenta con una nueva opción de ejecución:

- R, esta opción hace que Pound recargue las directivas *WafRules* de su fichero de configuración.

Las nuevas funciones, resultantes del desarrollo son las siguientes:

Definición de funciones

Fichero: *pound.h*

```
 extern void config_parse_waf(void);

 void waf_del_transaction(Transaction **transac);
```

```
void waf_create_transaction(Transaction **t, ModSecurity *ms, Rules
   *rules);

void waf_reload_rules(void);

int waf_body_enabled(int bodybuf, const char *logtag, int body_size,
   int chunked, int rpc);
int waf_add_http_info(Transaction *t, const char *header);

int waf_add_req_head(Transaction *t, char const **headers, int
   num_headers);
int read_body(BIO *sock, char **buff, int size);

int waf_resolution(Transaction *t,int *int_code, char **url);
```

Por último, el esquema modificado con los cambios añadido de la función *do_http*, encargada de gestionar la conexión de un cliente:

(CL <- LB) Abre el socket SSL con el cliente.
(CL -> LB) Obtiene las cabeceras del cliente.
(CL -> LB) Obtiene el body del cliente y lo almacena en un buffer.
Calcula servicio y backend al que debe dirigirse la petición.
(LB -> BK) Conecta con el backend elegido.
Crear una transacción WAF.
Inspecciona la petición del cliente.
(LB -> BK) Envía las cabeceras al backend.
(LB -> BK) Añade cabeceras de proxy.
~~(CL -> LB -> BK) Reenvía el body de cliente a backend.~~
(LB -> BK) Envía el body, almacenado en el buffer, al backend
(LB <- BK) Obtiene las cabeceras de la respuesta del backend.
(LB <- BK) Obtiene el body de la respuesta del backend.
Inspecciona la respuesta del backend.
(CL <- LB) Envia las cabeceras al cliente.
(CL <- LB) Añade cabeceras proxy al cliente.
~~(CL <- LB <- BK) Reenvía el body de backend a cliente.~~
(CL <- LB) Envía el body de respuesta al cliente, que estaba almacenado en el buffer.
Elimina la transacción WAF.
Log del resultado de la comunicación.
Si el protocolo es HTTP/1.1, se queda eperando una nueva petición.

Figura 12. Funcionamiento de la función "do_http" tras el desarrollo WAF.

5 API DE CONFIGURACIÓN

La finalidad de desarrollar una API, *Application Programming Interface*, es hacer el uso del WAF más intuitivo para aquellos usuarios que no estén familiarizados con la sintaxis SecLang. Uno de los requisitos de esta API es preservar la total compatibilidad con las sentencias SecLang, habiendo siempre la posibilidad de administrar las reglas WAF mediante la API o mediante sentencias SecLang propias de Libmodsecurity. También se permitirá importar ficheros de configuración de proveedores privados sin perder ninguna de sus características.

En todo momento esta API será un apoyo, pudiéndose configurar las propiedades más comunes de Libmodsecurity, aunque no se tendrá el uso total de SecLang. Si el administrador necesitase ir más allá de lo que la interfaz permite, deberá utilizar directamente directivas SecLang con las que detallar comportamientos más específicos.

5.1 Diseño de la API

El desarrollo de la API se ha hecho en lenguaje Perl. Perl es un lenguaje de *scripting,* de más alto nivel que otros lenguajes compilados como podría ser C. Perl está optimizado para la gestión de ficheros y la aplicación de expresiones regulares, por lo que lo hace ideal en la manipulación de archivos de configuración. Perl también destaca por la facilidad que permite a la hora de tratar datos, ya sea mediante *hashes* utilizados en el desarrollo para crear estructuras complejas, o *slices*, utilizados para gestionar listas.

Por otra parte, el modelo de programación usado es una arquitectura modelo-controlador. Esta arquitectura separa el código en dos partes:

- El modelo, es el encargado de la validación de los parámetros de entrada y la estandarización de los objetos de salida. El modelo no contiene ninguna lógica de aplicación.
- El controlador, realiza toda la lógica de la aplicación. En este caso es una librería consumida por el modelo y que trabaja sobre los ficheros de configuración del WAF.

La API cumple una estandarización RESTful pudiendo funcionar como un módulo de una aplicación mayor. Los verbos utilizados son:

- GET, para listar objetos o mostrar su contenido.
- POST, para crear objetos o aplicar acciones.
- PUT, para modificar objetos.
- DELETE, para eliminar objetos.

El ciclo que seguiría una petición a la API se detallará en este punto y cumple el siguiente esquema general:

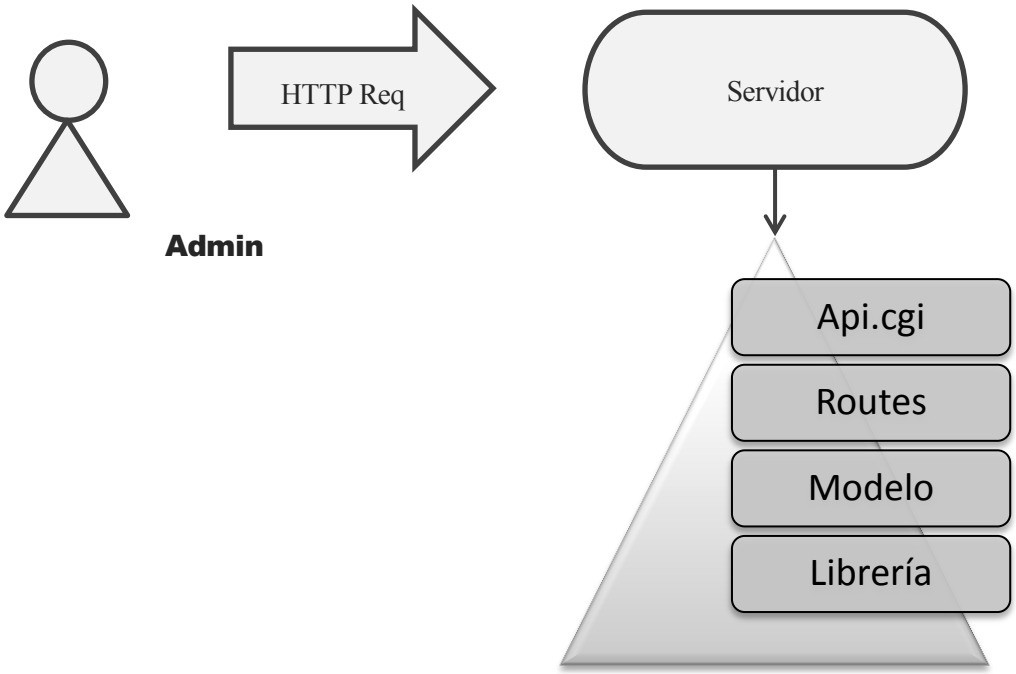

Figura 13. Arquitectura API

5.1.1 Servidor Web

La API estará servida por un servidor Web que se comunicará con ella por medio de CGI. También se requiere que el servidor pueda soportar todos los verbos HTTP usados por la API (GET, POST, PUT y DELETE). Se ha elegido Apache2 por ser un servidor conocido y versátil, que se encuentra en los repositorios de Debian y su instalación es inmediata.

```
$ sudo apt-get install apache2
```

Apache necesita activar la configuración CGI para poder servir aplicaciones basadas en esta tecnología. Ésto se consigue añadiendo un enlace simbólico desde las configuraciones disponibles a las configuraciones activas.

```
$ sudo ln –s /etc/apache2/mods-available/cgi.load  \
/etc/apache2/mods-enabled/cgi.load
```

A continuación se reescribirá la ruta desde la se cargan los CGI a la que se encuentra el script de la API.

Configuración de Apache

Fichero: *serve-cgi-bin.conf*

```
<IfModule mod_alias.c>
   <IfModule mod_cgi.c>
      Define ENABLE_USR_LIB_CGI_BIN
   </IfModule>

   <IfModule mod_cgid.c>
```

```
      Define ENABLE_USR_LIB_CGI_BIN
   </IfModule>

     <IfDefine ENABLE_USR_LIB_CGI_BIN>
        ScriptAlias / /opt/waf/api/
        <Directory "/opt/waf/api">
           AllowOverride None
           Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
           Require all granted
        </Directory>
     </IfDefine>
</IfModule>
```

Por último es necesario reiniciar el servidor para cargar la nueva configuración.

```
$ sudo /etc/init.d/apache2 restart
```

5.1.2 Script de ejecución

La comunicación entre el servidor y la aplicación se hará mediante CGI. El CGI será recogido por un script Perl que interpretará los parámetros HTTP más significativos como: método, URI y argumentos de peticiones POST y PUT. Estos parámetros se usan por el módulo rutas para indexar con la funcionalidad de la API demandada.

Una vez se realice la lógica necesaria mediante la librería, se escribirá en la salida estándar del programa el mensaje HTTP resultante, para que lo interprete el servidor Web y lo envíe al usuario.

El script utilizado en este proyecto, es un script más reducido y adaptado, obtenido de un proyecto de software libre llamado Zevenet. El script corresponde con el fichero "/usr/local/zevenet/www/zapi/v3.1/zapi.cgi" que se encuentra en el repositorio.

Script api.cgi

Fichero: *api.cgi*

```
#!/usr/bin/perl

use WAF::HTTP;
use CGI::Simple;

my $cgi = CGI::Simple->new();
my $input_ref;

if ($ENV{ CONTENT_TYPE } eq 'application/json' )
{
    my $data;
    $data = $cgi->param('POSTDATA')
       if $ENV{ REQUEST_METHOD } eq 'POST';
    $data = $cgi->param('PUTDATA')
       if $ENV{ REQUEST_METHOD } eq 'PUT';
```

```
    require JSON::XS;
    JSON::XS->import;

    $input_ref = eval { decode_json( $data ) };
}

##### Load API routes
require WAF::Routes;

my $desc = 'Request not found';
my $req = $ENV{ PATH_INFO };

&httpResponse( {code => 404, desc => $desc, msg => "$desc: $req"} );
```

El script se apoya en una función encargada de adaptar la respuesta a un formato CGI correcto.

```
sub httpResponse
{
  my $self = shift;

  # Headers included in the responses, any method, any URI,
  # success or error
  my @headers = (
    'Access-Control-Allow-Origin' => "http://$ENV{ HTTP_HOST }/",
    'Access-Control-Allow-Credentials' => 'true',
    'Cache-Control'                    => 'no-cache',
    'Expires'                          => '-1',
    'Pragma'                           => 'no-cache',
  );

  # header
  my $content_type = 'application/json';
  if $self->{ type } && $self->{ body }
  {
    $content_type = $self->{ type };
  }

  my $output = $q->header(
    -type    => $content_type,
    -charset => 'utf-8',
    -status  =>
        "$self->{ code } $http_status_codes{ $self->{ code } }",

    # extra headers
    @headers,
  );

  # body
  if ( exists $self->{ body } )
  {
     if ( ref $self->{ body } eq 'HASH' )
     {
```

```
        require JSON::XS;
          JSON::XS->import;

            my $json = JSON::XS->new->utf8->pretty( 1 );
            my $json_canonical = 1;
            $json->canonical( [$json_canonical] );

            $output .= $json->encode( $self->{ body } );
          }
          else
          {
            $output .= $self->{ body };
          }
      }

  print $output;
  exit;
}

1;
```

Para ejecutar este script se necesitan dos librerías de Perl extras:

- Libcgi-simple-perl, para el manejo de los CGI.
- Libjson-xs-perl, para el manejo de JSON.

Una vez creado el script es necesario aplicarle permisos de ejecución:

```
$ sudo chmod +x /opt/waf/api/api.cgi
```

5.1.3 Rutas

Buscando la escalabilidad de la aplicación y poder integrar la API con otros sistemas y futuros desarrollos. Se ha creado un módulo de enrutado, *Routes.pm*, que mediante las URIs y los métodos HTTP pueda seleccionar el módulo sobre el que actuar y la acción a realizar en él. Aquí se sigue una metodología REST, esta metodología utiliza los distintos verbos HTTP (POST, GET, PUT y DELETE) para aplicar acciones sobre objetos que se identifican mediante la URL.

Así por ejemplo, para borrar un conjunto de reglas WAF debe ejecutarse una petición HTTP como la siguiente:

```
DELETE http://127.0.0.1/waf/set-ejemplo
```

Para listar las reglas de un set debe utilizarse una llamada como:

```
GET http://127.0.0.1/waf/set-ejemplo
```

El objeto *set-ejemplo* corresponde con el nombre del objeto sobre el que estamos actuando.

Para extraer los identificadores de objetos *set* o *rule* se utilizarán expresiones regulares. Una vez identificada la

acción y el objeto sobre el que se va actuar, se enrutará con su modelo correspondiente.

Enrutado de peticiones API

Fichero: *Routes.pm*

```
# $cgi is a object with the cgi variables
# they are got by the Web server
if ( $cgi->path_info =~ qr{^/waf} )
{
  require API::Waf;

  # define regular expression
  my $set_re = '[\w-]+';

  # Get a WAF set
  if ( $cgi->{method} eq 'GET' && $cgi->{path} =~ '/waf/($set_re)$' )
  {
    &retrieve_set( $cgi->{params}, $1 );
  }

  # Create a WAF set
  if ( $cgi->{method} eq 'POST' && $cgi->{path} =~ '/waf$' )
  {
    &create_set( $cgi->{params} );
  }
```

5.1.4 Modelo

El modelo controla la entrada y salida de datos de la API. Es una etapa previa al tratamiento de los datos, en la que se validan las entradas y se comprueban los posibles conflictos que pueden existir entre parámetros. En la salida, se estandarizan las respuestas, añadiendo códigos HTTP con los que identificar el resultado de la petición, y se estandarizan los formatos de datos propios de cada parámetro del objeto.

Las comprobaciones que se aplican antes de ejecutar una acción son:

- Si el objeto sobre el que se va a aplicar la acción existe.
- Si todos los parámetros requeridos han sido enviados.
- Si algún parámetro no es correcto para la llamada que se está ejecutando.
- Si el valor de algún parámetro no es correcto.
- Si existe ya un objeto antes de crear un objeto con ese nombre.
- Si ya se encuentra aplicado un *set* a una granja, antes de aplicarlo.

Modelo de una petición API

Fichero: *Set.pm*, función: *create_waf_set()*

```
#  POST /waf
```

```
sub create_waf_set
{
   my $json_obj = shift;

   require 'WAF::Config';

   my $desc = "Create the WAF set, $json_obj->{ 'name' }";

   # Check if it exists
   if ( &existWAFSet( $json_obj->{ 'name' } ) )
   {
      my $msg = "$json_obj->{ 'name' } already exists.";
      return &httpErrorResponse( code => 400, desc => $desc,
         msg => $msg );
   }

   my $params = {
      "name" => {
         'valid_format' => 'waf_set_name',
         'non_blank'    => 'true',
         'required'     => 'true',
      },
      "copy_from" => {
         'valid_format' => 'waf_set_name',
         'non_blank'    => 'true',
      },
   };

   # Check allowed parameters
   my $error_msg = &checkAPIParams( $json_obj, $params );
   return &httpErrorResponse( code => 400, desc => $desc,
      msg => $error_msg ) if ( $error_msg );

   # executing the action
   if ( exists $json_obj->{ 'copy_from' } )
   {
      unless ( &existWAFSet( $json_obj->{ 'copy_from' } ) )
      {
         my $msg = "$json_obj->{ 'copy_from' } does not exist.";
         return &httpErrorResponse( code => 404, desc => $desc,
            msg => $msg );
      }

      &copyWAFSet( $json_obj->{ 'name' },
         $json_obj->{ 'copy_from' } );
   }
   else
   {
      &createWAFSet( $json_obj->{ 'name' } );
   }

   my $output = &getZapiWAFSet( $json_obj->{ 'name' } );

   # check result and return success or failure
   if ( $output )
   {
      my $msg = "Added the WAF set $json_obj->{ 'name' }";
```

```
      my $body = {
         description => $desc,
         params      => $output,
         message     => $msg,
      };
      return &httpResponse( { code => 201, body => $body } );
   }
   else
   {
      my $msg =
         "Error, trying to create the WAF set $json_obj->{ name }";
      return &httpErrorResponse( code => 400, desc => $desc,
         msg => $msg );
   }
}
```

Para asegurar que la API siempre devuelve los mismos parámetros y garantizar los formatos de cada uno de los campos se han estandarizado las salidas de la API a través del fichero *Struct.pm*. Un ejemplo de la estandarización de los parámetros de una regla WAF es el siguiente:

Estandarización de parámetros de salida en la API

Fichero: *Struct.pm*, función: *getZapiWAFRule()*

```
sub getZapiWAFRule
{
    my $rule = shift;

    require 'WAF::Core';
    my $out;

    if ( $rule->{ type } =~ /(?:match_action|action)/ )
    {
        $out = {
           'type'            => $rule->{ type }            // '',
           'rule_id'         => $rule->{ rule_id } + 0     // '',
           'description'     => $rule->{ description }     // '',
           'tag'             => $rule->{ tag }             // [],
           'severity'        => $rule->{ severity }        // '',
           'phase'           => $rule->{ phase }           // '',
           'transformations' => $rule->{ transformations } // [],
           'multi_match'     => $rule->{ multi_match }     // '',
           'capture'         => $rule->{ capture }         // '',
           'action'          => $rule->{ action }          // '',
           'log'             => $rule->{ log }             // '',
           'audit'           => $rule->{ audit_log }       // '',
           'log_data'        => $rule->{ log_data }        // '',
           'set_variable'    => $rule->{ set_variable }    // [],
           'chain'           => $rule->{ chain }           // [],
           'skip'            => $rule->{ skip }            // '',
           'skip_after'      => $rule->{ skip_after }      // '',
           'http_code'       => $rule->{ http_code }       // '',
           'execute'         => $rule->{ execute }         // '',
        };
```

5.1.5 Librería

La librería se integrará con las librerías de Perl, para ello utilizarán el *path* nativo en el que el lenguaje busca sus módulos, */usr/share/perl5/*. Aquí se ha creado un enlace símbólico a la localización original del código "*/opt/waf*". Estas funciones son las encargadas de la gestión del WAF y tienen la funcionalidad de controlador en la arquitectura planteada, modelo-controlador. Se han separado en 4 archivos en función de su contexto:

- *Core.pm*. Son las funciones básicas que no dependen de otras. Son sólo de lectura.
- *Parser.pm*. Son las funciones encargadas de parsear y construir los ficheros de configuración de Libmodsecurity.
- *Config.pm*. Son las funciones que trabajan con objetos de Perl, operando sobre ellos y modificando sus parámetros. Luego serán transcritos a ficheros de configuración a través de *Parser.pm*.
- *Runtime.pm*. Son las funciones que ejecutan binarios en el sistema.

A parte de los ficheros propios de la librería, en esta ruta está el ya nombrado *Routes.pm* y un subdirectorio con el modelo de la API.

5.2 Modelo de datos

El modelo de datos descrito en este punto es en el cual se basarán los objetos usados por Perl. Este modelo hace las funciones de *wrapper* para SecLang y permite mayor facilidad para crear y modificar las reglas.

Se crearán dos objetos, uno llamado *rule* que corresponde a una directiva SecLang, y otro llamado *set* en el cual se agruparán reglas y se les podrá dotar de una configuración global para todo el conjunto.

Los objetos construidos se basarán en las directivas, variables, operadores y acciones de SecLang, explicadas en el punto 1.3.1 de la memoria.

5.2.1 Rules

Una regla corresponde a una directiva. Se permitirá crear 4 tipos de reglas diferentes.

- *Mark*, son marcas a las que se podrá saltar con el objetivo que evitar las reglas anteriores. Estas reglas corresponden a la directiva *SecMarker*.
- *Actions*, aplican acciones incondicionalmente. Son usadas principalmente para inicializar variables o crear opciones por defecto. Corresponden con la directiva *SecAction*.
- *Match-action*, aplican una acción si se cumple un patrón definido en el *match*. Corresponde con la directiva *SecRule*.
- *Custom,* son las directivas que no correspondan a ninguna de las anteriores. El valor de este tipo de reglas es una cadena con la regla en formato SecLang.

A continuación se detallarán los posibles campos resultantes de parsear una directiva SecLang.

Los campos de la tabla siguiente son compartidos por todos los tipos de reglas:

Identificador	Tipo de dato	Descripción
Id	Numérico	Posición de la regla dentro del set
Type	String	Tipo de regla: mark, custom, action o match-action
Raw	String	Regla en formato SecLang. Tal como se encuentra en el fichero de configuración

Tabla 1. Campos comunes para todas las reglas

Las reglas de tipo *mark* añaden el campo:

Identificador	Tipo de dato	Descripción
Mark	String	Nombre con el que referenciar la marca.

Tabla 2. Campos específicos para reglas 'mark'

Dentro de las reglas *match-action* y *action* pende otro objeto llamado *chain*. Los *chains* o cadenas corresponden a la acción *chain* de una directiva *SecRule* o *SecAction*. Estas acciones hacen que la siguiente regla sólo se ejecute en el caso de que la primera haya tenido éxito. Sirven para crear patrones más complejos.

En cambio, las reglas *action* y *match-action* añaden los campos:

Identificador	Tipo de dato	Descripción
Rule_id	Numérico	Identificador único para una regla.
Description	String	Mensaje descriptivo.
Tag	Lista de Strings	Etiquetas para clasificar las reglas.
Version	Numérico	Versión mínima de Libmodsecurity para que la regla funcione.
Madurity	Numérico	Nivel de madurez que tiene la regla.
Severity	Numérico o string	Nivel de criticidad de la regla.
Accuracy	Numérico	Nivel de exactitud de la regla.
Revision	Numérico	Número de revisión de la regla.
Phase	Numérico o String	Fase de libmodsecurity en la que se aplicará la regla.
Transformations	Lista de Strings	Transformaciones que se le aplican a la variable antes de examinar un patrón.
Multi_match	String	Examina el patrón para cada una de las transformaciones.
Capture	String	Captura datos de expresiones regulares.
Action	String	Acción a aplicar cuando el patrón detecte una coincidencia.

Http_code	Numérico	Código HTTP para las acciones *redirect* o *deny*
Modify_directive	Lista de Strings	Modifica una directiva en función de que se cumpla el patrón.
Execute	String	Ruta a un script LUA que se ejecutará.
No_log	String	Fuerza que la regla no se escriba en el fichero de logs.
Log	String	Escribe en el log en caso de que la regla se cumpla.
Audit_log	String	Fuerza que la regla no se escriba en el log de auditoría.
No_audit_log	String	Escribe en el log de auditoría en caso de que la regla se cumpla.
Init_colection	Lista de Strings	Inicializa una colección de variables.
Set_uid	String	Asigna un valor a la variable USERID.
Set_sid	String	Asigna un valor a la variable SESSION.
Set_variable	Lista de Strings.	Aplica una acción de modificar variable, por cada directiva setvar que aparezca en la regla SecLang.
Expire_variable	Lista de Strings.	Asigna un tiempo tras el cual una variable es borrada.
Chain	Lista de reglas encadenadas.	Son reglas encadenadas, que se ejecutarán de forma secuencial.
Skip	Numérico	Salta un número de reglas.
Skip_after	String o numérico	Salta un número de reglas o hasta una marca.

Tabla 3. Campos para reglas 'action' y ‘match-action'

Las los campos específicos de las reglas *match-action* son los que implementan el *match*. Combinando estos tres campos se pueden crear patrones de búsqueda en el tráfico.

Identificador	**Tipo de dato**	**Descripción**
Variables	Lista de Strings	Variables para analizar con el operador. Vienen definidas en la documentación de Libmodsecurity
Operator	String	Acción a aplicar para buscar patrones en cada una de las variables. Vienen expecificadas en la documentación de Libmodsecurity.
Operating	String	Es el valor que se le aplica al operador para buscar en las variables.

Tabla 4. Campos específicos de reglas 'match-action'

Un ejemplo de los campos anteriores reflejados sobre una regla sería:

```
SecRule &TX:crs_exclusions_wordpress|TX:crs_exclusions_wordpress \
    "@eq 0" \
    "id:9002000,\
    phase:1,\
    pass,\
    t:none,\
    nolog,\
    skipAfter:END-WORDPRESS"
```

Variables: &TX:crs_exclusions_wordpress y TX:crs_exclusions_wordpress

Operator: eq

Operating: 0

Id: 9002000

Phase: 1

Action: pass

Transformations: none

No_log: true

Skip_after: END-WORDPRESS

5.2.2 Sets

Los sets son agrupaciones de reglas que cumplen un contexto común. A través del *set* será posible añadirle una configuración común a todas las reglas que lo forman. Esta configuración consiste una representación visual a las directivas de configuración más usadas. Los parámetros de configuración podrán extenderse con facilidad, debido a la escalabilidad que permiten los *hashes* de Perl, en los que se basan.

Los objetos *set* a su vez, están compuestos por dos campos: una lista de reglas y una configuración. La lista de reglas es una lista de objetos ordenados que refleja el orden en que aparecen en el fichero de configuración, que es el orden con el que se ejecutan. Las directivas del fichero no tienen porqué corresponderse estrictamente con los objetos de reglas listados, ya que las reglas que tengan definido el parámetro *chain* serán anidadas en una sóla regla.

El campo configuración del *set*, que define el comportamiento general del conjunto, está formado de los siguientes parámetros:

Identificador	**Tipo de dato**	**Descripción**
Default_action	String	Establece una acción por defecto para el campo *action* de una regla.
Default_log	String	Establece una acción por defecto para el campo *log* de una regla.
Default_phase	String	Establece una acción por defecto para el campo *phase* de una regla.

Audit	String	Habilita o deshabilita el log de auditoría.
Process_request_body	String	Habilita o deshabilita examinar el body de las peticiones.
Process_response_body	String	Habilita o deshabilita examinar el body de las respuestas.
Request_body_limit	Numérico	Establece un tamaño máximo para el body de la petición.
Status	String	Habilita o deshabilita el *set*. También puede establecerse el modo solo detección.
Disable_rules	Lista de números	Deshabilita las reglas especificadas en la lista. Se utiliza el campo *rule_id* de la regla.

Tabla 5. Campos de configuración de un set.

5.3 Lectura y escritura de los ficheros configuración WAF

Los objetos *set* son las representaciones de ficheros de configuración con directivas SecLang y que seguirán el siguiente formato:

```
## begin conf
SecDefaultAction "allow,phase:1,nolog"
## end conf


SecRule REQUEST_METHOD "@streq POST" " \
   id:9001184,\
   phase:1,\
   t:none,\
   pass,\
   nolog,\
   noauditlog,\
   chain"

SecRule REQUEST_FILENAME "@rx /file/ajax/field_asset_[a-z0-9_]$" "\
   chain"

SecMarker mark_testing #
```

El primer bloque definido entre las marcas "*## begin conf*" y "*##end conf*" es la configuración del *set* .El resto de reglas que aparecen después son las reglas contenidas en el *set*.

Para convertir las reglas en objetos y viceversa, se ha definido las siguientes funciones:

- Parse_set y build_set.
- Parse_conf y build_conf.
- Parse_rule y build_rule.
- Parse_operator, parse_variable y parse_action.
- Build_batch.

Las funciones *parse* se utilizarán para las lecturas de los ficheros de configuración, mientras que las *build* se usarán para escrituras. Las funciones *parse* obtienen los parámetros de las reglas y *sets* por medio de expresiones regulares, aplicando una expresión por cada parámetro que se desea buscar.

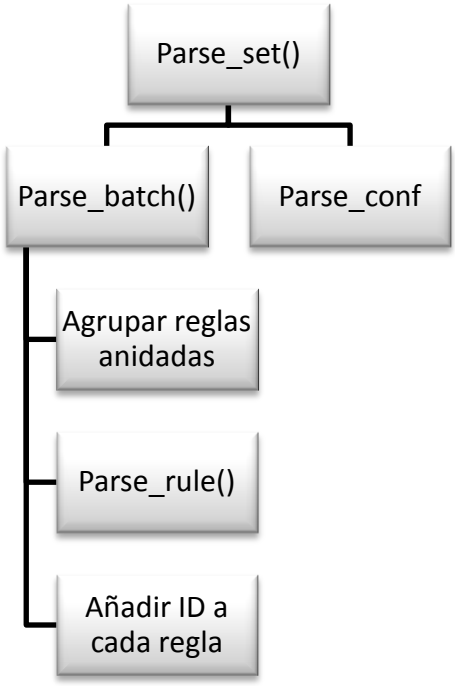

Figura 14. Esquema para obtener un objeto *set* de reglas

La función parse_set, responsable de formar el objeto set a través de las directivas SecLang de un fichero, se apoya principalmente en la función *parse_rule*. Esta función es una función recursiva, que se llamará a sí misma para autocontener cada uno de los *chains* que penden de la regla principal. El esquema que cumple es el siguiente:

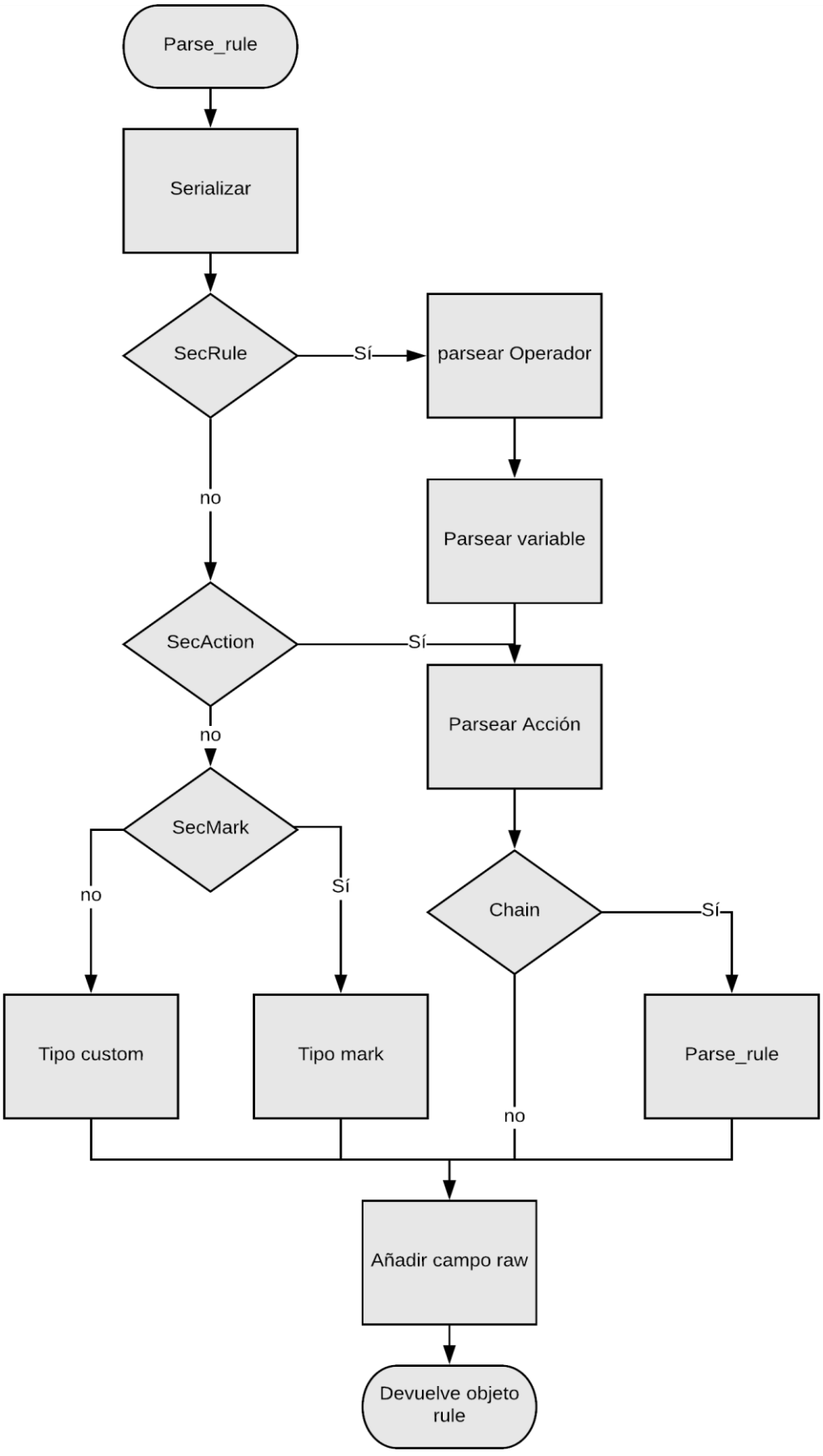

Figura 15. Esquema de la función "parse_rule"

A su vez *parse_rule* utiliza las funciones *parse_operator* y *parse_variables* para obtener los parámetros exclusivos de las reglas *match-action*:

Parse_rule

```
my $rule;

if ( $line =~
   /^\s*SecRule\s+"?([^"]+)"?\s+"?([^"]+)"?\s+"?([^"]+)?"?/s )
{
   my $var = $1;
   my $ope = $2;
   my $act = $3;

   &parseWAFVariables($rule, $var);
   &parseWAFOperator($rule, $ope);
   &parseWAFAction($rule, $act);
}
elsif ( $line =~ /^\s*SecAction\s+"?([^"]+)?"?/s )
{
   my $act = $3;
   &parseWAFAction($rule, $act);
}
```

Parse_variables

```
sub getWAFVariables
{
    my $rule = shift;
    my $var = shift;
    my @var_sp = split ( '\|', $var_reg );
    $rule->{variables} = \@var_sp;
}
```

Parse_operator

```
sub getWAFOperator
{
    my $rule = shift;
    my $oper = shift;
    my $operator = "";
    my $operating = "";

    if ( $oper =~ /^(?<operator>!?\@\w+)?\s+?(?<operating>[^"]+)?$/ )
    {
        $rule->{operator} = $+{ operator } // "";
        $rule->{operating} = $+{ operating };
```

```
    }

    # set not_operator
    if ( $rule->{operator} )
    {
        $rule->{operator} =~ s/^(!)?\@//;
        my $not_op = $1 // '';
        $rule->{operator} = "${not_op}$rule->{operator}";
    }
}
```

Parse_actions

```
sub getWAFActions
{
   my $rule = shift;
   my $act = shift;

   my @options = split ( ',', $act );
   foreach my $param ( @options )
   {
      $param =~ s/^\s*//;
      $param =~ s/\s*$//;

      if ( $param =~ /msg:'?([^']+)'?/ )
      {
         $rule->{ description } = $1;
      }
      elsif ( $param =~ /id:'?([^']+)'?/ )
      {
         $rule->{ rule_id } = $1;
      }
      elsif ( $param =~ /tag:'?([^']+)'?/ )
      {
         push @{ $rule->{ tag } }, $1;}
      }
# ...
      elsif ( $param =~ /^chain$/ )
      {
         foreach my $ru ( @nested_rules )
         {
            $rule->{ raw } .= "\n" . &convertWAFLine( $ru );
            push @{ $rule->{ chain } }, &parseWAFRule( $ru );
            $rule->{ chain }->[-1]->{ type } = 'match_action';
         }
      }
# ...
```

Para transcribir un *set* en un fichero se utilizará la función *build_set*. La parte de escritura es más simple, *build_conf* crea las directivas correspondientes a la configuración y a continuación se llama a *build_rule* una vez por cada una de las reglas que hay definidas en la lista de reglas del *set*. Las reglas se escribirán usando

una línea por cada uno de los parámetros que la regla tenga configurados, añadiendo el carácter "\" al final de línea para mejorar la legibilidad.

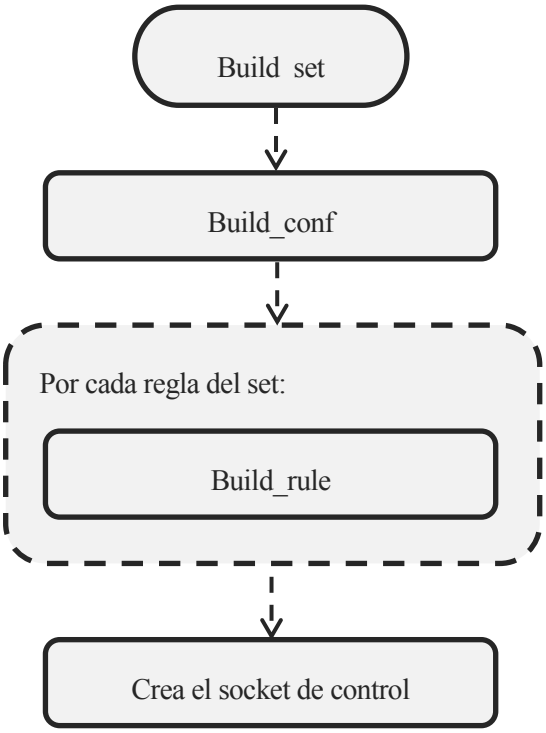

Figura 16. Esquema de la función "build_set"

Antes de sobreescribir el set se comprobará que comprobará su sintaxis, asegurando que ninguna de las reglas está mal escrita o que existe alguna incompatibilidad que rompa el set. Para ello se utilizará la opción '-W' que se añadió a Pound para comprobar la sintaxis de las de un fichero de reglas WAF.

Check_syntax

```
sub checkWAFFileSyntax
{
   my $file = shift;

   my $out   = `pound -W $file 2>&1`;
   my $err   = $?;

   if ( $err )
   {
      chomp $out;

      #parse response and return field that failed
      my @aux = split ( '\n', $out );
      $out = $aux[1];
      $out =~ s/^.+Column: \d+. //;
   }
   else
```

```
    {
       $out = "";
    }
    return $out;
}
```

Por último se recargarán los procesos de Pound que estuviesen utilizando dicho *set* de reglas. Aquí se utilizará la opción '-R' de Pound.

Reload_set

```
sub reloadWAFByFarm
{
     my $farm = shift;
     my $err  = 0;

     require WAF::Parser;

     my $socket    = getFarmSocket( $farm );
     my $set_file;

     # check set
     foreach my $set ( &listWAFByFarm( $farm ) )
     {
          $set_file = &getWAFSetFile( $set );
          return 1 if ( &checkWAFFileSyntax( $set_file ) );
     }

     $err = system( "pound_ctl -c $socket -R" );

     return $err;
}
```

A veces interesará modificar reglas SecLang sin utilizar los objetos que se han definido. Ya sea porque se requiera modificar una directiva o parámetro de los que no soportan estos objetos, o se quieran añadir reglas SecLang que se hayan obtenido de alguna fuente. Para ello se utiliza el parámetro *raw* definido en todas las reglas. En este parámetro puede ser enviada una o varias directivas. Sólo es necesario crear una cadena con las reglas y separarlas por un carácter de fin de línea '\n'. Para tratar el campo *raw* se ha creado la función *build_batch*, que sustituye una regla ya existente o añade una nueva al *set* con el contenido de este campo. Una vez la regla ha sido añadida al *set*, éste se comprobará para comprobar que la nueva regla no rompe la coherencia del *set*.

5.4 Aplicar reglas al balanceador

Un balanceador de carga define como granja a la agrupación de un conjunto de servidores, que se encuentran sirviendo el mismo servicio, bajo una misma interfaz virtual del balanceador. Para los ejemplos aquí mostrados, una granja corresponde a un proceso de Pound que arranca con una configuración y sirve un servicio a través de una IP y puerto virtuales.

Una vez haya configurado uno o varios *sets* será posible asociarlos con procesos de Pound con el objetivo de que el tráfico HTTP pueda ser analizado. En esta memoria se utilizará el directorio "*/opt/farms*" para guardar los archivos de configuración de Pound, cada uno de ellos representa una granja de balanceo.

Para trabajar la asociación de un *set* y una granja se añadirán tres llamadas para la API, las cuales implementarán las acciones: añadir un *set* a la granja, eliminar un *set* de una granja y cambiar la posición de un *set* dentro de una granja. Los esquemas para cada una de las acciones son:

Para mover los *sets* se ha utilizado una de las propiedades que tiene Perl, los *slices*. Los *slices* es la forma que usa Perl de manejar listas, y es que Perl permite manipular las listas como si fuesen subconjuntos de datos, así de una forma ágil pueden reescribirse listas reordenándolas durante el proceso.

```
sub moveByIndex
{
        my ( $list, $ori_index, $dst_index ) = @_;

        my $elem = $list->[ $ori_index ];

        # delete item
        splice (@{ $list }, $ori_index, 1);

        # add item
        splice (@{ $list }, $dst_index, 0, $elem);
}
```

5.5 Resumen

5.5.1 Ficheros

Tras el desarrollo de la API el árbol de ficheros de código y rutas de configuración es el siguiente:

```
/opt/waf
|
|--api/
|   |
|   |--api.cgi
|
|--config/
|
|--Routes.pm
|
|--Model/
|   |
|   |--Sets.pm
|   |
|   |--Rules.pm
|   |
|   |--Structs.pm
```

```
|
|--lib/
   |
   |--Actions.pm
   |
   |--Core.pm
   |
   |--Parser.pm
   |
   |--Config.pm

/opt/farms
```

Los directorios lib y Model y el fichero Routes.pm tienen un enlace simbólico en el directorio "/usr/share/perl5/WAF/" para poder ser utilizados como un módulo más de Perl.

Los ficheros más relevantes del servidor Web son:

- /etc/apache2/mods-enabled/cgi.load. Habilitar CGI
- /etc/apache2/conf-available/serve-cgi-bin.conf. Configuración de CGI
- /etc/init.d/apache2. Script de parada y arranque de apache

5.5.2 Objetos

Los objetos definidos en el apartado 3.2, utilizados por la librería que gestiona el WAF, no corresponden estrictamente a los objetos que aparecen en la interfaz de la API. Esto es debido a que los campos menos usados se han ocultado y otros se han agrupado, con la finalidad de hacer una herramienta más simple de manejar.

El objeto *set* contiene los siguientes campos:

Identificador	Tipo de dato	Descripción
Default_action	String	Establece una acción por defecto para el campo *action* de una regla.
Default_log	String	Establece una acción por defecto para el campo *log* de una regla.
Default_phase	String	Establece una acción por defecto para el campo *phase* de una regla.
Audit	String	Habilita o deshabilita el log de auditoría.
Process_request_body	String	Habilita o deshabilita examinar el body de las peticiones.
Process_response_body	String	Habilita o deshabilita examinar el body de las respuestas.
Request_body_limit	Numérico	Establece un tamaño máximo para el body de la petición.

Status	String	Habilita o deshabilita el *set*. También puede establecerse el modo solo detección.
Disable_rules	Lista de números	Deshabilita las reglas especificadas en la lista. Se utiliza el campo *rule_id* de la regla.

Tabla 6. Campos de configuración de un set API

El objeto regla contiene los siguientes campos:

(los códigos que corresponden a la segunda columna "Reglas que lo soportan" son ; c: custom, m: mark, a: action, -: match-action y x: chain).

(los códigos que corresponden a la tercera columna "Tipo de campo" son ; d: descriptivo, a: acción o m: match).

Identificador	**Reglas que lo soportan**					**Tipo de campo**			**Descripción**
	c	**m**	**a**	**-**	**x**	**d**	**m**	**a**	
Id	X	X	X	X		X			Posición de la regla dentro del set
Type	X	X	X	X		X			Tipo de regla: mark, custom, action o match-action
Raw	X	X	X	X					Regla en formato SecLang. Tal como se encuentra en el fichero de configuración
Mark		X						X	Nombre con el que referenciar la marca.
Variables				X	X		X		Variables para analizar con el operador. Vienen definidas en la documentación de Libmodsecurity
Operator				X	X		X		Acción a aplicar para buscar patrones en cada una de las variables. Vienen expecificadas en la documentación de Libmodsecurity.
Operating				X	X		X		Es el valor que se le aplica al operador para buscar en las variables.
Rule_id			X	X		X			Identificador único para una regla.
Description			X	X		X			Mensaje descriptivo.
Tag			X	X		X			Etiquetas para clasificar las reglas.
Severity			X	X		X			Nivel de criticidad de la regla.
Phase			X	X			X		Fase de libmodsecurity en la que se aplicará la regla.
Transformations			X	X	X		X		Transformaciones que se le aplican a la variable antes de examinar un patrón.

Multi_match		X	X	X	X	Examina el patrón para cada una de las transformaciones.
Capture		X	X	X	X	Captura datos de expresiones regulares.
Action		X	X		X	Acción a aplicar cuando el patrón detecte una coincidencia.
Http_code		X	X		X	Código HTTP para las acciones *redirect* o *deny*
Execute		X	X	X	X	Ruta a un script LUA que se ejecutará.
Log		X	X		X	Escribe en el log en caso de que la regla se cumpla.
Audit		X	X		X	Fuerza que la regla no se escriba en el log de auditoría.
Set_variable		X	X	X	X	Aplica una acción de modificar variable, por cada directiva setvar que aparezca en la regla SecLang.
Chain			X		X	Son reglas encadenadas, que se ejecutarán de forma secuencial.
Skip		X	X		X	Salta un número de reglas.
Skip_after		X	X		X	Salta un número de reglas o hasta una marca.

Tabla 7. Tabla de parámetros para cada tipo de regla

5.5.3 Acciones

Los parámetros de la URI que aparecen entre los caracteres '<' y '>' identifican a objetos que deben haber sido creados y sobre los que se realizará la acción.

Método	**URI**	**Descripción**
GET	/waf	Lista los sets disponibles.
POST	/waf	Crea un nuevo set.
GET	/waf/<set>	Detalla los parámetros de un set
PUT	/waf/<set>	Modifica la configuración global del set.
DELETE	/waf/<set>	Elimina un set.
POST	/waf/<set>/rules	Crea una nueva regla en un set.
GET	/waf/<set>/rules/<rule>	Detalla una regla determinada de un set.
PUT	/waf/<set>/rules/<rule>	Modifica los parámetros de una regla.

	/waf/<set>/rules/<rule>	Elimina una regla de un set.
POST	/waf/<set>/rules/<rule>/actions	Aplica una acción a una regla. La única acción disponible es cambiar la posición de la regla dentro del set.
POST	/waf/<set>/rules/<rule>/chains	Crea un objeto chain en una regla. Esta acción sólo está disponible para reglas *match-action*
PUT	/waf/<set>/rules/<rule>/chains/<chain>	Modifica un objeto chain de una regla. Esta acción sólo está disponible para reglas *match-action*
DELETE	/waf/<set>/rules/<rule>/chains/<chain>	Elimina un objeto chain de una regla. Esta acción sólo está disponible para reglas *match-action*
POST	/farms/<farm>/waf	Añade un set a una granja.
DELETE	/farms/<farm>/waf/<set>	Elimina un set de una granja.
POST	/farms/<farm>/waf/<set>/actions	Cambia la posición de un set dentro de una granja.

Tabla 8. Tabla de peticiones de la API

5.5.4 Códigos HTTP de respuesta

Código	OK/Error	Descripción
200	OK	La petición resultó exitosa.
201	OK	La petición creó un nuevo objeto de forma correcta.
400	Error	Error en los parámetros de entrada.
400	Error	Error, el objeto que se desea crear ya existe.
400	Error	Error genérico. La acción no pudo realizarse correctamente.
404	Error	Error, el objeto sobre el que se desea actuar no existe.
404	Error	Error, la petición (combinación de URI y método) no existe.
500	Error	Error en el servidor. El proceso murió durante la ejecución.

Tabla 9. Tabla de códigos HTTP para la API

5.5.5 Crear una regla

Utilizando la tabla del punto 3.2.2 del objeto reglas es posible crear reglas de forma intuitiva como si de otro firewall se tratase.

Los campos descriptivos son meramente informativos, y no intervienen en el match y no tienen una repercusión sobre las acciones. Éstos son: *type*, *rule_id*, *tag*, *severity*, *id*, *description*.

La secuencia a seguir para crear el match de la regla sería:

- El campo *fase* indica en qué momento del procesamiento se realizará el *match*.
- La *variable* es dónde debe aplicarse el *operador*.
- La *transformación* es usada en aquellos casos en los que la *variable* pueda venir codificada o se desee transpolar a otro valor antes de aplicarle el *operador*.
- El *multi operador* permite comprobar el *operador* para cada una de las *transformaciones* que se van aplicando sobre la *variable*.
- El *operador* aplica la operación seleccionada sobre la *variable*.
- Es frecuente que el *operador* necesite un valor con el que operar. Este valor es el campo *Operando*.
- En caso de que el *operador* pueda capturar un valor, como en el caso de expresiones regulares, si se desea capturar, hay que indicarlo en la variable de *captura*.
- Por último si es necesario encadenar varios matches, habría que usar el objeto *chain* que comparte la campos con el *match*.

Match:

Campo	Phase	Variable	Transformation	Multi_op	Operator	Operating	Capture
Posible valor	Head_req	ARGS	Base64decode	☐	detectSQL	<valor>	☐
	Body_req	FILES	Md5		detectXSS		
	Head_resp	COOKIES	Lowercase		ipMatch		
	Body_resp	SERVER_ADDR	HexDecode		rx		
	Logging	SESSION	removeNulls		eq		
		...	....		...		

Tabla 10. Posibles valores de un match.

Para definir la acción a realizar en caso de que el *match* sea exitoso se utilizan los campos de tipo acción.

- El campo *acción* implica una decisión de permitir o cesar la conexión HTTP.
- El campo *set_variable* permite cambiar el valor de una variable.
- El campo *execute* ejecuta un script.
- Los campos *Log, Audit* y *Log_data* aplican acciones de registro en logs.
- Los campos *skip* y *skip_after* alteran el flujo del análisis de reglas saltando a otra regla.

Las reglas más sencillas que pueden ser aplicadas se componen de: *variable*, *operador*, *operando*, y una *acción*. Un ejemplo de una regla podría ser bloquear las peticiones que no sean GET a una aplicación Web. Para crear una regla con ese criterio habría que mandar una petición como:

```
curl -X POST -H 'Content-Type: application/json' \
```

```
d'{"variables":["REQUEST_METHOD"],"operator":"!streq", \
"operating":"GET","action":"deny"}' \
http://192.168.100.10/api.cgi/waf/PRUEBAS/rules
```

Si se compara la aplicación de una regla con otro firewall, como puede ser el caso de IPtables, se puede ver que la sintaxis es similar:

```
Iptables –A \
–destine 192.168.0.0/16 –j REJECT
```

En el caso de iptables, la acción REJECT implica un fin de conexión en el caso de que se cumpla el patrón previo. El patrón está formado por la "variable" IP de destino y el "operating" 192.168.0.0/16. El "operator" en este caso no es posible especificarlo ya que obligatoriamente es una comprobación de máscara de red

6 VALIDACIÓN

"Fiarse de todo y no fiarse de nada son dos vicios. Pero en el uno se encuentra más virtud, y en el otro, más seguridad." – Lucio Anneo Séneca

6.1 Pruebas de rendimiento, Benchmark

Las pruebas que se realizarán han sido basadas en los tests de rendimiento oficiales, expuestas por los desarrolladores de Libmodsecurity en la *wiki* de su proyecto. Se ha hecho una serie de cambios en las pruebas como es la máquina en la que corre el *software*, y la arquitectura de la red, ya que en las pruebas oficiales la librería Libmodsecurity corre directamente sobre el servidor Web y aquí correrá sobre el balanceador de carga.

6.1.1 Esquema del benchmark

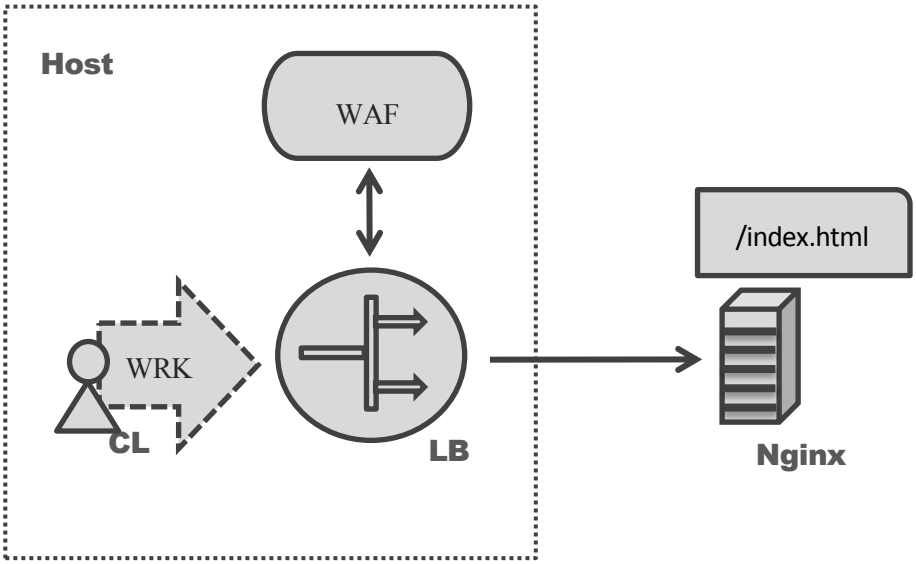

Figura 17. Escenario de pruebas de rendimiento

Para seguir con el esquema de benchmark usado por los desarrolladores de Libmodsecurity, tanto el cliente como el proxy estarán ejecutados en la misma máquina. El sistema se compone de los siguientes elementos:

- Como cliente de peticiones se ha elegido un software llamado *wrk*, éste software permite realizar peticiones HTTP en paralelo contra un servidor Web. Está escrito en C y usa una metodología por eventos. Su rendimiento es bastante bueno. El comando lanzado para todas las pruebas será el mismo, pudiendo ver la diferencia de rendimiento y peticiones por segundo conforme se modifique la configuración del WAF. El comando será:

  ```
  wrk -c 20 -d 20 -t 10 http://127.0.0.1/index.html
  ```

- Para proxy HTTP, encargado de mandar las peticiones a la librería WAF y posteriormente al servidor, usaremos dos implementaciones con el objetivo de comparar su rendimiento. En primer lugar usaremos la explicada a lo largo de esta memoria, Pound; en segundo lugar, usaremos Nginx como proxy HTTP, compilado con el conector de Libmodsecurity que ha desarrollado el mismo equipo de Libmodsecurity.
- La versión de la librería Libmodsecurity usada es la 3.0.2.
- Como servidor Web se ha elegido nginx, ya que ofrece mucho más rendimiento que Apache, y no queremos que el servidor Web se convierta en un cuello de botella en el sistema.

Característica	
CPU	Intel Xeon E3-1245 v5, 3.5GHz con 8 cores
Memoria	8GB DDR4 2133MHZ ECC

Tabla 11. Características de la máquina anfitriona de las pruebas de rendimiento

6.1.2 Las configuraciones

Se ejecutará una vez el benchmark para cada una de las siguientes configuraciones y ambos proxys, Pound y Nginx.

- Sin aplicar ninguna configuración WAF.
- Aplicando el set *ruleset-light*, es la configuración recomendada, sin ninguna directiva.
- Aplicando una sola directiva.
- Aplicando 10 directivas.
- Aplicando 100 directivas.
- Aplicando el *set* de directivas de OWASP.
- Aplicando 500 directivas de analizar las cabeceras de petición.
- Aplicando 500 directivas de analizar las cabeceras de respuesta.
- Aplicando 600 directivas de analizar las cabeceras de petición.
- Aplicando 600 directivas de analizar las cabeceras de respuesta

6.1.3 Resultados

Configuración	peticiones/seg Pound		peticiones/seg Nginx		Pound/Nginx
	Req/sec	Eficiencia	Req/sec	Eficiencia	
Proxy sin WAF	120000		45180		2.65
ruleset-light	36514	30.43%	30793	68.16%	1.18
1 directiva	52275	43.56%	39378	87.16%	1.32
10 directivas	43357	36.13%	36060	79.81%	1.20
100 directivas	18068	15.06%	17197	38.06%	1.05
owasp-ruleset	2667	2.22%	2613	5.78%	1.02
500 directivas - req head	4795	4.00%	4945	10.95%	0.96

500 directivas - resp head	2091	1.74%	2200	4.87%	0.95
600 directivas - req head	4042	3.37%	4225	9.35%	0.95

Tabla 12. Resultados de pruebas de rendimiento

Los cuadros rojos indican que la CPU está cerca de saturar, por encima del 90%.

Las columnas de Pound y Nginx se subdividen en dos columnas a su vez. La primera es el número de peticiones HTTP por segundo que el sistema ha procesado. La segunda es la eficiencia del sistema con respecto al primer caso, el sistema sin procesar WAF.

La última columna refleja una comparativa del rendimiento entre la implementación hecha con Pound y la hecha con Nginx.

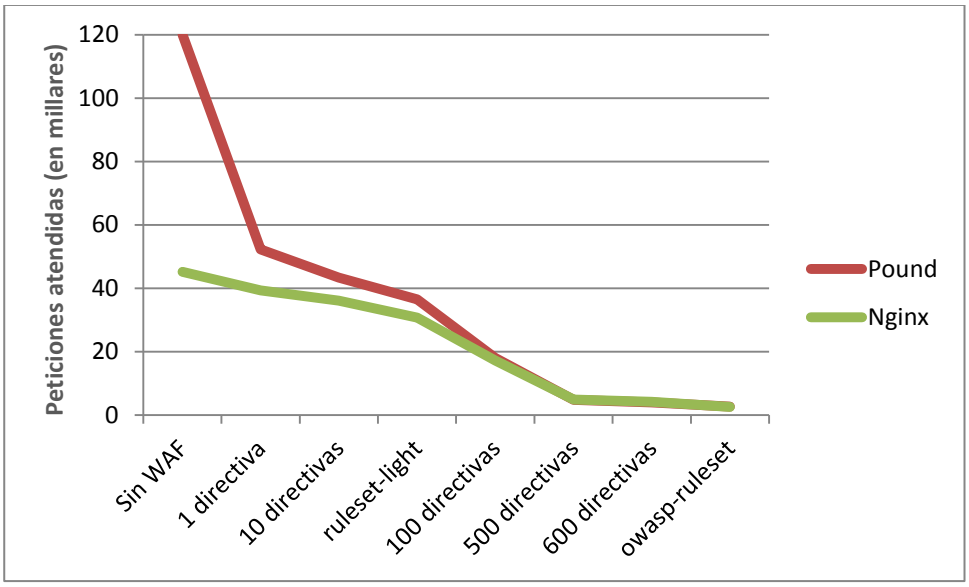

Figura 18 . Gráfico de peticiones respecto al número de directivas analizadas

6.2 Tests de la API

Debido a que el proyecto expuesto a lo largo de esta memoria consiste en un desarrollo *software*, se ha querido dotar a la implementación de unos tests funcionales que aseguren la integración continua del código en futuros desarrollos y faciliten su depuración. Estos tests consistirán en peticiones HTTP en las cuales se pruebe la API de configuración, se examine que las respuestas sean coherentes y el estado de los procesos sea el correcto. Cada test será recursivo, esto quiere decir que una vez finalizado el test, el sistema quedará en el mismo estado en el que se encontraba antes de probarlo.

Se han definido dos tests, cada uno de ellos escritos en Perl.

- En un primer test se comprobará la configuración, se lanzarán todas las llamadas creadas en el punto 3 y se probará cada uno de los parámetros de cada tipo de objetos WAF.
- En un segundo test se creará un sistema de balanceo funcional, se crearán reglas de protección y se realizarán peticiones HTTP contra el balanceador, comprobando tanto peticiones que deben ser bloqueadas como que deben ser permitidas.

El árbol en el cual se organizan los ficheros pertenecientes a los tests siguen el siguiente esquema:

```
|
|--tests/
|       |
|       |--waf_config.t
|       |
|       |--waf_requests.t
|
|--Includes/
|       |
|       |--Objets/
|       |        |
|       |        |--waf.pm
|       |
|       |--Api/
|                |
|                |--waf.pm
|
|--Functions.pm
|
|--Config.ini
```

6.2.1 Configuración

El fichero *Config.ini* contiene la configuración relativa al balanceador de carga. Esta información es usada por las funciones de test tanto para lanzar peticiones como para conectarse por SSH y comprobar cosas en el sistema.

```
IP=192.168.100.150
pass=admin1admin
```

6.2.2 Definiciones

En el directorio *includes* aparecen las definiciones de objetos importados por los tests. Se han creado dos tipos de includes:

- Los ficheros del directorio *objects* son definiciones de objetos reglas y *sets*, cada parámetro del objeto tiene valores establecido. El sentido de este fichero es crear objetos funcionales y útiles que puedan ser importados en cualquier test, modularizando la aplicación y facilitando su uso.

Objects/waf.pm

```
Sub obj_waf_set_config
{
  return {
      'audit'                 => 'true',
      'process_request_body'  => 'true',
      'process_response_body' => 'true',
      'request_body_limit'    => '6456456',
      'disable_rules'         => [100,102],
```

```
      'default_action'        => 'pass',
      'default_log'           => 'true',
      'default_phase'         => 2,
      'status'                => 'detection',
  };
}
```

- Los ficheros del directorio *api* son las definiciones de las llamadas a la API. Aquí se guardan estructuras con los campos necesarios para hacer cada una de las llamadas a la API. Por ejemplo que verbo usa, a que URI hay que efectuar la llamada o qué parámetros está esperando. Es posible cargarlas usando objetos reglas y set, en ese caso la llamada API se parametrizaría con los datos necesarios para crear, modificar… ese objetos en concreto.

Api/waf.pm

```
sub list_set
{
  return {
    'method' => 'GET',
    'uri'    => "/waf",
  };
}


# create_waf ( $set_obj )
sub create_waf
{
  my $params = shift;
  my $set = $params->{ name };

  return {
    'method' => 'POST',
    'uri'    => "/waf",
    'params' => {
      'name' => $set,
    },
    'remote_checks' => [
      {
        "cmd" => "ls $WAF_PATH/${set}.conf",
      },
    ],
  };
}
```

6.2.3 Funciones

Toda la lógica usada en los tests está en el fichero *Functions.pm*. La función más relevante es la que ejecuta cada una de las sentencias test, que utiliza el módulo *Test::More* para procesar la resolución.

La función *test* espera una lista de acciones, las cuales ejecutará y comprobará su salida de error. La función

test a su vez se apoya en 3 funciones, cada una de ellas ejecuta un tipo diferente de comprobación.

La función ***Api*** ejecuta una llamada a la API utilizando los parámetros de los campos '*method*', '*uri*' y '*params*'. Una vez ejecutada la llamada se comprueba el código de error, aceptándose 2xx como éxito. Posteriormente la función *test*, quién lanza *api*, devuelve el objeto JSON respondido por la API, por si se quisiesen hacer comprobaciones extra sobre los parámetros devueltos por la aplicación.

Parámetros de entrada para la función *api.*

```
{
   'method' => 'PUT',
   'uri' => "/set/$name",
   'params' => {
      'ip'      => $ip,
       'netmask'    => $netmask,
       'gateway'  => $gateway,
   },
}
```

La función *api* realiza la petición HTTP usando el cliente de la librería *LWP::UserAgent*. La creación de la petición se compone de las siguientes etapas:

- Inicialización de la interfaz por la que se comunica con el servidor Web:

```
$ua = LWP::UserAgent->new(
  agent    => '',
  ssl_opts => {
    verify_hostname => 0,
    SSL_verify_mode => 0x00
  }
);
```

- Formulación de la petición.

```
my $request = getRequest( $method, $uri );
```

- Si la petición es POST o PUT, se crea el body del mensaje, añadiéndose los argumentos y la cabecera *content-type* correspondiente. Los parámetros serán codificados en formato JSON con el módulo de *JSON*.

```
if ( $method eq 'POST' or $method eq 'PUT' )
{
  $request->content_type( 'application/json' );
  $request->content( JSON::encode_json( $params ) );
}
```

- Una vez creada la petición, se ejecuta y se obtiene la salida.

```
my $response = $ua->request( $request );
```

- Por último se devuelve el código HTTP resultante de la petición y el objeto JSON con los argumentos de respuesta decodificados.

```
eval { $json_dec = JSON::decode_json( $response->content() ); };
return $out = {
  'code'     => $response->code,
  'Params'   => $json_dec,
};
```

Durante el test también pueden ejecutar **comandos en el nodo remoto**, el balanceador de carga sobre el que se están realizando las pruebas, con la intención de conocer el estado del sistema. Ésto es útil para saber si las acciones se están aplicando correctamente y si el estado de los procesos es el deseado. Las comprobaciones remotas son comandos linux que se ejecutan mediante SSH. Estas comprobaciones tienen dos parámetros, el primero es el comando que se desea ejecutar, y el segundo es el resultado esperado tras la ejecución.

Definición de una comprobación remota:

```
'remote_checks' = [
  {
    "cmd"        : "groups user | grep rbac",
    "expect_err" : "true",
  }
];
```

Ejecución y comprobación del comando:

```
$cmd_out = system("ssh root\@$HOST $cmd" );

if ( $cmd_out and $expect_err eq 'false' )
{
  $error += 1;
}
elsif ( !$cmd_out and $expect_err eq 'true' )
{
  $error += 1;
}
```

Los **comandos locales** son ejecutados en la máquina cliente, la que ejecuta los tests contra el balanceador de carga. El interés de estas comprobaciones es lanzar aplicaciones de cliente para extraer datos como: el rendimiento o el estado de un servicio. Al igual que en las comprobaciones remotas los parámetros usados son: el comando a ejecutar, y si se espera algún error de la ejecución.

Definición de una comprobación local:

```
'local_checks' = [
  {
    "cmd"     => "groups user | grep rbac",
    "expect_err" : "true",
  }
];
```

Ejecución y comprobación del comando:

```
$cmd_out = system( $cmd );

if ( $cmd_out and $expect_err eq 'false' )
{
  $error += 1;
}
elsif ( !$cmd_out and $expect_err eq 'true' )
{
  $error += 1;
}
```

6.2.4 Tests

Los ficheros tests son scripts en Perl, se encuentran en el directorio *tests* y están identificados por la extensión .t. Son pruebas que se ejecutan de forma secuencial y en las que se aplican acciones sobre el balanceador de carga para posteriormente comprobar su efecto. Cada una de las pruebas es una ejecución de la función test ya antes detallada.

Las partes de las que se compone un tests son:

- Carga dependencias y variables de configuración:

```
#!/usr/bin/perl
use Config::Tiny;
require Functions;

# VARIABLES
my $Config = Config::Tiny->read( 'config.ini' );
my $HOST         = $Config->{ _ }{ HOST };
```

- Carga de objetos:

```
# define objects
my $mark        = &obj_waf_mark();
my $rule1       = &obj_waf_rule_all();
my $rule2       = &obj_waf_rule_all_2();
my $set_conf    = &obj_waf_set_conf_all();

my $setname1 = 'test_waf_ori';

my $test_def;
my $error = 0;
my $api;
```

- Carga de llamada API y ejecución:

```
# Creating a set
$test_def = &create_waf_rule( $rule1, [$setname1] );
$error += &test( $test_def );

# creating a rule
$test_def = &custom_waf_rule( $custom, [$setname1] );
$error += &test( $test_def );

# showing a rule
$test_def = &show_waf_rule( [$setname1, 0] );
$error += &test( $test_def );

# copying a rule
$test_def = &copy_waf_rule( { copy_from => $rule1->{ rule_id } },
  [$setname1] );
$error += &test( $test_def );
```

Para validar el desarrollo API se han creado dos test:

- Test de configuración. Ejecuta todas las llamadas de la API y configura un sistema WAF completo, evaluando gran cantidad de opciones.

- Test funcional. Crea un sistema funcional, configurando una granja de balanceo, añadiéndole un backend y aplicándole una configuración de seguridad. Tras esto ejecuta peticiones HTTP y se comprueba que sean bloqueadas.

6.2.5 Ejecución

Para ejecutar los tests, se utilizará el la herramienta *prove* de Linux ya que lo test han sido integrados con el módulo *Test::More* que se integra con la herramienta.

```
cd tests
prove -I. waf_conf.t
```

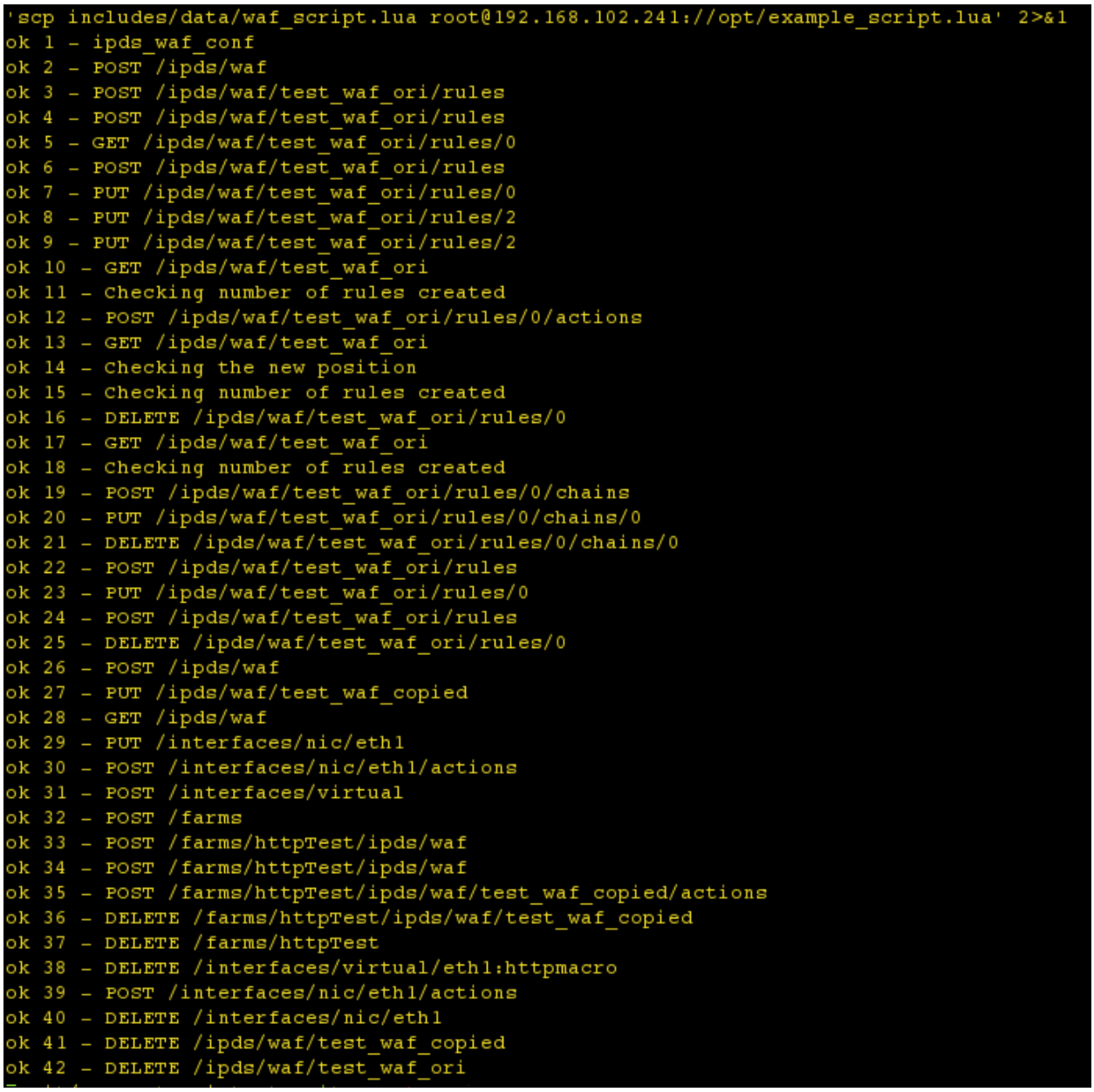

Figura 19. Resultado del test de configuración

```
prove -I. waf_requests.t
```

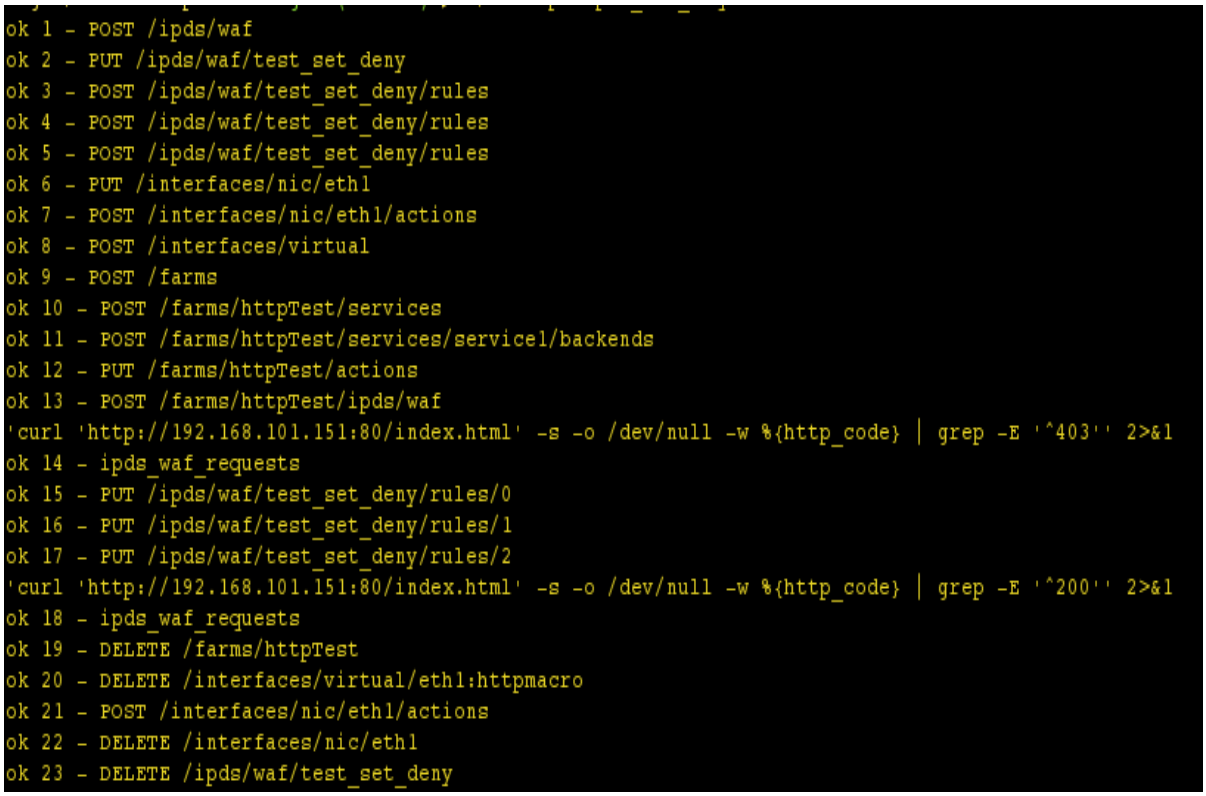

Figura 20. Resultado del test funcional de sistema

6.3 Usos prácticos

En este punto se realizarán peticiones HTTP que simulan algunos de los ataques a Webs más comunes. El tráfico circulará a través del balanceador de carga, que será el encargado de examinarlo y bloquear aquellas peticiones que intenten aprovecharse de alguna de las vulnerabilidades. Las reglas WAF se aplicarán mediante la API, utilizando la interfaz definida en el capítulo 3 de la memoria.

6.3.1 Escenario de pruebas

HOST	IP	Puerto
Admin	192.168.	-
CL	192.168.1.186	-
LB	192.168.102.241	8080
API	192.168.102.241	80
BK	192.168.100.127	80

Tabla 13. Hosts del escenario “usos prácticos”

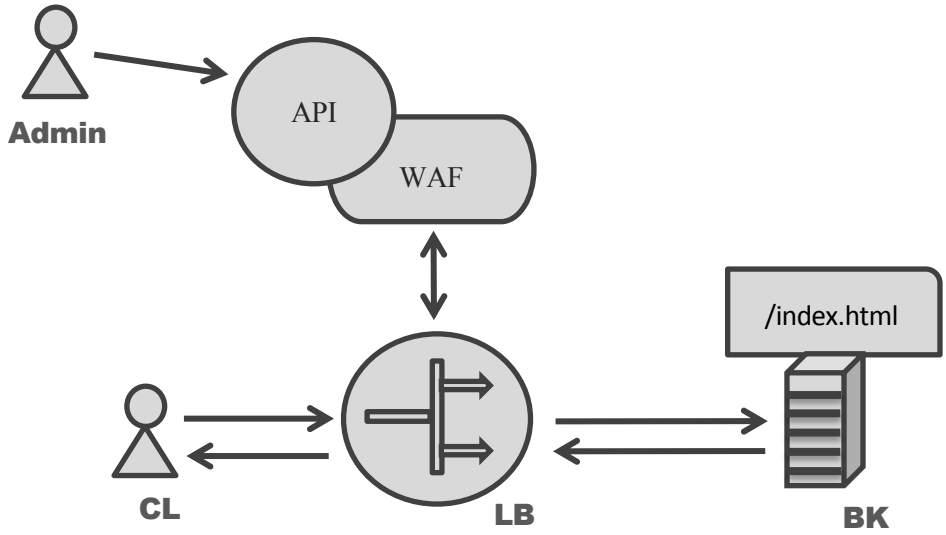

Figura 21. Escenario para usos prácticos

El backend mostrado en el escenario devuelve una página con un simple mensaje "servidor alcanzado" y un código 200 con lo que comprobar si la petición ha sido respondida por el backend. El balanceador de carga, bloqueará la petición y devolverá un código 403 al cliente en caso de que alguna regla WAF aplique.

La configuración con la que Pound arranca es la siguiente:

```
Name lb_granja
ThreadModel  dynamic
Control  "/tmp/lb_control.socket"

ListenHTTP
  Address 192.168.102.241
  Port 8080
  Service "servers"
    BackEnd
      Address 1192.168.100.127
      Port 80
    End
  End
End
```

Para hacer funcionar estas pruebas es importante que Libmodsecurity haya compilado con la librería YAJL (encargada de parser objetos JSON), ya que se utilizarán objetos JSON para enviar datos en las peticiones POST. También hacen falta algunas reglas que nos aseguren que los JSON son decodificados, estas reglas se encuentran en el *set* de reglas recomendadas por los desarrolladores de Libmodsecurity. Por ello se copiará dicho set en el *path* usado para las reglas WAF, una vez copiado, la API podrá trabajar con él como si hubiese sido creado por un usuario. El fichero de reglas recomendadas puede encontrarse dentro del directorio principal del proyecto.

Copia el set y se asigna a la granja de balanceo

```
scp modsecurity.conf-recommended \
  root@192.168.102.241:/opt/waf/config/recommended_set.conf

curl -X POST -H 'Content-Type: application/json' -d \
  '{"name":"recommended_set"}' \
  http://192.168.102.241:80/api.cgi/farms/lb_granja/waf
```

Una vez todas las máquinas están configuradas conforme al esquema, se gestionará mediante la API la creación de los sets de reglas, la asignación con la granja y algunas configuraciones necesarias.

Crear un set de reglas

```
curl -X POST -H 'Content-Type: application/json' -d \
'{"name":"custom_set"}' \
http://192.168.102.241:80/api.cgi/waf
```

Asignar el set creado a una granja

```
curl -X POST -H 'Content-Type: application/json' -d \
  '{"name":"custom_set"}' \
  http://192.168.102.241:80/api.cgi/farms/lb_granja/waf
```

Habilitar que el WAF pueda bloquear tráfico

```
curl -X PUT -H 'Content-Type: application/json' -d \
  '{"status":"true"}' \
  http://192.168.102.241:80/api.cgi/waf/custom_set
```

Para crear una regla WAF se utilizará un comando curl como el siguiente, en el que únicamente variará el objeto JSON enviado en el cuerpo de la petición, indicado con el parámetro ***–d***.

Petición genérica para crear una regla WAF.

```
curl -X POST -H 'Content-Type: application/json' -d \
  'OBJ_JSON' \
  http://192.168.102.241:80/api.cgi/waf/custom_set/rules
```

6.3.2 Validación entradas

Es muy frecuente que los desarrolladores no validen los parámetros de entrada de una aplicación, ésto puede conllevar a inyección de código malicioso o fallos en la lógica de la aplicación si algún argumento tiene un valor no esperado.

Para ello vamos a forzar que un parámetro de entrada "status" tenga un valor exclusivo de *true* o *false*. No se especificará si el parámetro llegara por método GET o POST, comprobándose ambos

Objeto Json enviado.

```
 {
   "action" : "deny",
   "phase":"2",
   "variables" :[
     "ARGS:json.status"
   ],
   "operator" : "!strPhrases",
   "operating":"true false"
}
```

Regla SecLang del fichero de configuración.

```
 SecRule ARGS:json.status "!@pm true false" "\
         id:3,\
         phase:2,\
        deny" #
```

Petición HTTP contra el backend

```
 curl --request POST \
   --url http://192.168.102.241:8080/ \
   --header 'content-type: application/json' \
   --writeout "\n\nhttp code: %{http_code}"
   --data '{
     "status":"blank"
    }'
```

Respuesta HTTP

```
Replied forbidden

Error code: 403
```

Otra de las posibilidades que existen es que el argumento de entrada deba cumplir un patrón, expresado por una expresión regular.

Objeto Json enviado.

```
 {
   "action" : "deny",
   "phase":"2",
   "variables" :[
     "ARGS:json.name"
   ],
   "operator" : "!strRegex",
   "operating" : "\\w-"
}
```

Regla SecLang del fichero de configuración.

```
 SecRule ARGS:json.name "!@rx \w-" "\
         id:4,\
         phase:2,\
        deny" #
```

Esta característica es muy útil a la hora de parchear vulnerabilidades relacionadas con las entradas, sin tener que esperar los tiempos de trabajo (desarrollo, testing y puesta en producción) de una actualización para la aplicación Web.

6.3.3 Fuga de información

Uno de los problemas con más repercusión es la fuga de información. Ésto puede poner al descubierto datos de clientes (tarjeta de crédito, contraseña, identidad...) o datos de valor para la empresa (facturas, contenido a la venta...). Puede deberse a varios motivos, directorios que quedan expuestos, backups, ataques a la base de datos...

En este apartado se verificará la respuesta enviada por el servidor, para analizar si se está enviando información sensible. Para ello, se ha habilitado una nueva URL en la que hay alojado un fichero de texto plano, que responderá cuando se pida dicha URL. El texto simulará el contenido de una bases de datos, que va a ser entregada al cliente tras explotar una vulnerabilidad. Para detectar que se está enviando información sensible, se introducirá en el fichero un campo nuevo que representa una entrada en la base de datos, este nuevo campo hará te *trigger* para detectar la fuga.

BBDD.

```
 Carlos     Cano    Pérez      45821658V  647552847
 Francisco  Carmona González   25468698T  954789871
 Maria      Heredia Sánchez    69712053P  641587215
 Carmen     Blanco  Velázquez  27014982N  665231680
 triggervCpkA9WEDZ43V3WEasvDuCA null null null null
```

La última entrada del fichero es la que se ha utilizado como *trigger* y por tanto la que se utilizará para crear las reglas WAF.

Objeto Json enviado.

```
 {
  "action" : "deny",
   "phase":"4",
   "variables" :[
     "RESPONSE_BODY"
   ],
   "operator" : "strContainsWord",
   "operating" : "triggervCpkA9WEDZ43V3WEasvDuCA"
}
```

Regla SecLang del fichero de configuración.

```
 SecRule RESPONSE_BODY "@rx triggervCpkA9WEDZ43V3WEasvDuCA" "\
         id:5,\
         phase:4,\
         deny" #
```

Petición HTTP contra el backend

```
 curl --request GET \
  --url http://192.168.102.241:8080/bbdd.html
```

Respuesta HTTP

```
Replied forbidden

Error code: 403
```

Si el servidor debido a una mala configuración estuviera exponiendo algún directorio sensible, se podría restringir también su acceso desde el WAF. Por ejemplo si se quisiera bloquear el directorio *backup*:

Objeto Json enviado.

```
 {
  "action" : "deny",
   "phase":"1",
   "variables" :[
     "REQUEST_URI"
   ],
   "operator" : "strRegex",
```

```
   "operating" : "backup"
}
```

Regla SecLang del fichero de configuración.

```
SecRule REQUEST_URI "@rx backup" "\
        id:2,\
        phase:1,\
        deny" #
```

Petición HTTP contra el backend

```
curl --request GET \
 --url http://192.168.102.241:8080/backup/backup1.tar
```

Respuesta HTTP

```
Replied forbidden

Error code: 403
```

6.3.4 Inyección SQL

Los ataques de inyección SQL son ataques contra la base de datos, en los cuales los formularios no son comprobados y es posible engañar a la aplicación. Principalmente son usados como método para saltarse el login o obtener información directamente de la base de datos.

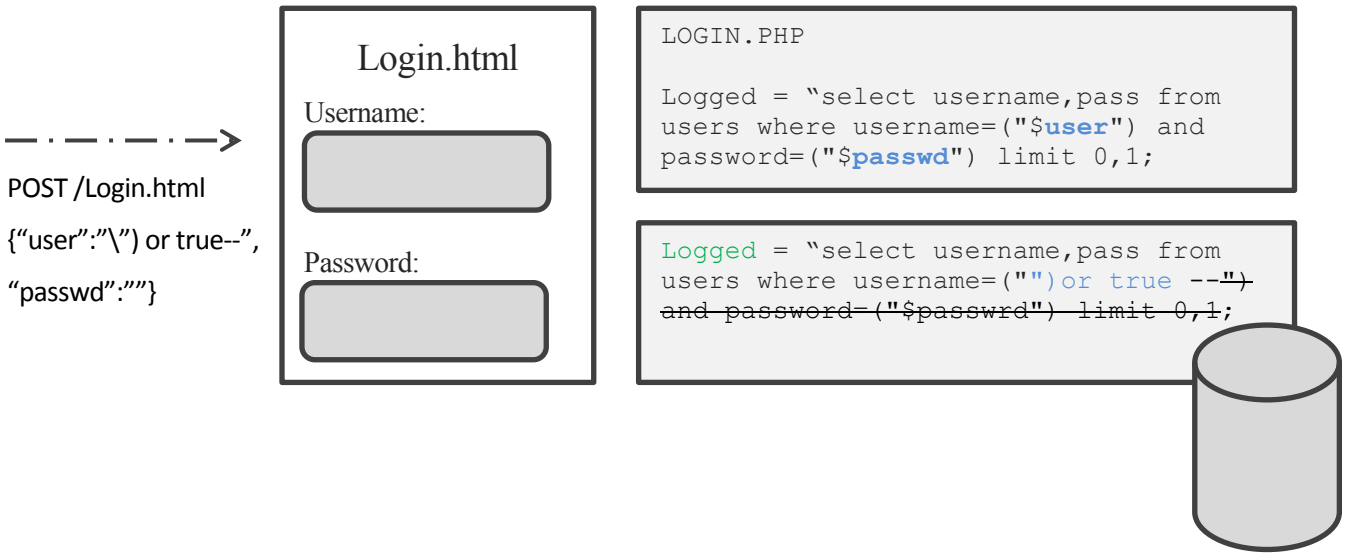

Figura 22. Esquema SQL injection

Libmodsecurity dispone de un operator dedicado apara este tipo de ataques, detectSQLi. Los campos más propensos a este tipo de ataques, son formularios en los que introducir el usuario y contraseña, por ello serán en los que se base esta prueba. Hay muchas combinaciones probables para hacer un ataque SQLi, cada una de ellas será efectiva o no dependiendo de la implementación que haya hecha en el servidor.

Objeto Json enviado.

```
{
  "username": "admin",
  "password":"\") or true--"
}
```

Otra posibilidad para la variable es generalizar el parámetro usuario y contraseña.

```
ARGS:(json)?.pass(word|wd)?|ARGS:(json)?.user(name)?
```

Regla SecLang del fichero de configuración.

```
SecRule ARGS.json.password "@detectSQLi " "\
        id:3,\
        phase:2,\
        deny" #
```

Petición HTTP contra el backend

```
curl --request POST \
  --url http://192.168.102.241:80
/ \
  --header 'content-type: application/json' \
```

```
  --data '{
    "username": "admin",
    "password":"\") or true--"
}'
```

Respuesta HTTP

```
Replied forbidden

Error code: 403
```

6.3.5 DoS

Algunas veces una aplicación contiene llamadas pesadas que pueden bloquear el servidor Web. Con el WAF se puede limitar el número de llamadas a un recurso. Para ello se utilizarán tres reglas con la siguiente secuencia:

- Registrar el recurso al que se está accediendo
- Bloquear si el contador ya ha llegado al límite.
- En caso de que no haya llegado, incrementar el contador.

Debido a que algunas de las propiedades necesarias para estas reglas no han sido implementadas en la API, se formularán las reglas a partir de la opción *raw*.

Objeto Json enviado.

```
{
 "raw" : [
  "SecAction \"phase:1,initcol:resource=%{REQUEST_URI},pass,nolog\"",
  "SecRule RESOURCE:COUNTER \"@gt 2\" \\",
  "\"deny,phase:1,id:7\"",
  "SecRule REQUEST_URI \"limited_resource\" \\",
  "\" phase:1,setvar:resource.counter=+1,pass,id:9\""
 ]
}
```

Regla SecLang del fichero de configuración.

```
SecAction "phase:1,initcol:resource=%{REQUEST_URI},pass,nolog" #


SecRule RESOURCE:COUNTER "@gt 2" \
"deny,phase:1,id:7" #
```

```
SecRule REQUEST_URI "limited_resource" \
"phase:1,setvar:resource.counter=+1,pass,id:9" #
```

La regla añadida bloqueará el recurso tras 3 consultas exitosas.

Petición HTTP contra el backend

```
curl --request GET \
 --url http://192.168.102.241:80/limited_resource.html
```

Respuestas HTTP

```
Try 1: [OK] Web alcanzada
Try 2: [OK] Web alcanzada
Try 3: [OK] Web alcanzada
Try 4: Replied forbidden
```

6.3.6 Ataques de fuerza bruta

Para proteger frente a ataques de fuerza bruta se combinarán técnicas ya vistas como el contador de intentos visto en DoS, la comprobación de parámetros de entrada y examinar el código de respuesta enviado por el servidor para saber el logging resultó fallido.

La regla que se aplicará bloqueará las peticiones de un cliente tras 3 intentos de contraseña incorrectos.

Objeto Json enviado.

```
{
 "raw" : [
  "SecAction \"initcol:ip=%{REMOTE_HOST},pass,nolog\"",
  "SecRule IP:COUNTER \"@gt 2\" \\",
  "\"deny,phase:4,id:7\"",
  "SecRule ARGS_NAMES.json \"password\" \\",
  "\"id:9,chain\"",
  "SecRule RESPONSE_STATUS \"^[45]\" \\",
  "\"phase:4,setvar:ip.counter=+1,pass\""
 ]
}
```

Regla SecLang del fichero de configuración.

```
SecAction "initcol:ip=%{REMOTE_HOST},pass,nolog"

SecRule IP:COUNTER "@gt 2" \
"deny,phase:4,id:7"
```

```
SecRule ARGS_NAMES.json "password" \
"id:9,chain"

SecRule RESPONSE_STATUS "^[45]" \
"phase:4,setvar:ip.counter=+1,pass"
```

Petición HTTP contra el backend

```
curl --request POST \
  --url http://192.168.102.241:80/ \
  --header 'content-type: application/json' \
  --data '{
    "username": "admin",
    "password":"pas1234"
  }'
```

Respuestas HTTP

```
Try 1: [OK] Web alcanzada
Try 2: [OK] Web alcanzada
Try 3: [OK] Web alcanzada
Try 4: Replied forbidden
```

7 CONCLUSIONES Y FUTUROS DESARROLLOS

"Nunca te das cuenta de lo que has logrado; sólo puedes ver lo que queda por hacer." - Marie Curie

Tras la etapa de validación se analizará el rendimiento y la gestión del WAF obteniendo algunas conclusiones. También, se dará una visión de cómo podría crecer el proyecto y qué futuros desarrollos sería posible acometer.

7.1 Conclusiones

El resultado del proyecto es un sistema totalmente funcional, formado por un balanceador de cargas, que inspecciona datos de las aplicaciones que virtualiza, y que es capaz de permitir o denegar peticiones en función de las reglas con las que se le configure.

Las reglas de seguridad del balanceador son configuradas a través de una API de gestión que se ha creado para dicha tarea. La API abstrae al operador de seguridad de la compleja sintaxis de reglas que utiliza la librería y permite crear objetos que busquen contenido potencialmente malicioso en los campos deseados de una trama HTTP. La API es una aplicación independiente que ha sido montada sobre un servidor Web en el mismo host que el balanceador y que construye reglas complejas a través de objetos simplificados.

Desde el punto de vista del rendimiento, y analizando los datos del apartado 6.1 puede verse que se ha obtenido un rendimiento similar al de otro producto ya en el mercado, alcanzando como máximo una penalización del 5% cuando el sistema se encuentra saturado. Con esto se entiende que el cuello de botella en la tecnología es la librería, que debido a la cantidad de comprobaciones por paquete que se debe hacer, se eleva el consumo de CPU.

Para aumentar el número de peticiones por segundo sería necesario aumentar la CPU de la máquina, por lo que en entornos industriales, con mejores *hardwares,* se podrían conseguir mejores *benchmarks*. De aquí se puede sacar un claro ejemplo de que la seguridad no es gratis, requiere de recursos de computación que al final repercute en equipos más caros.

El análisis de rendimiento se utilizó para saber los puntos de mejora del proyecto aunque desde un primer momento la solución no está enfocadas para centros de alto procesamiento. Es una solución compacta de balanceador de carga más capa de seguridad pensada para entornos como los de mediana empresa.

Con respecto a la gestión la herramienta API, creada para la gestión del WAF, es muy flexible, incluso puede paliar algunos de los ataques a servidores Web más comunes sólo filtrando el dato HTTP que se desea analizar. Aunque requiera de cierta familiarización con el lenguaje para construir reglas más complejas, se permite la importación de reglas en formato SecLang, haciendo la API totalmente compatible.

Para aquellos entornos con fuertes directivas de seguridad, la API permite abstraer al operador de seguridad de la infraestructura, no necesitando acceso a la máquina para poder gestionar la seguridad, y quedando los distintos entornos aislado.

Es importante recordar que las reglas requieren un mantenimiento, y actualizaciones en caso de que la aplicación cambie o aparezcan nuevas vulnerabilidades. La eficiencia de dichas reglas dependerá de los conocimientos del operador a la hora de construir reglas y del número de reglas que se examinen durante el análisis del tráfico. Siempre existiría la posibilidad de adquirir reglas de proveedores privados que ya se encargan de optimizar su funcionamiento y su precisión.

7.2 Futuros desarrollos

Tras finalizar el proyecto quedan abiertas nuevas tareas con las que dotarlo de nuevas propiedades y complementarlo con herramientas que lo hagan más atractivo de cara al usuario.

La siguiente etapa de desarrollo podría consistir en un interfaz gráfico, que de forma intuitiva facilite al usuario la creación de reglas sin recurrir directamente a la API. El entorno gráfico utilizaría la API como interfaz de comunicación con el WAF, presentando las tareas de mantenimiento sobre un formulario en el que se muestren los posibles valores para cada uno de los campos. Para crear el interfaz gráfico se podrían usar los modernos *frameworks* de desarrollo Web como Angular o React ya bien acogidos por la comunidad de desarrolladores. Tras implementar la parte gráfica, el proyecto tendría una estructura: vista (interfaz gráfica), modelo (API) y controlador (librería).

También una de las tendencias tecnológicas más de moda es la inteligencia artificial y machine learning. Por ello se podría crear una aplicación que de forma autónoma cree reglas optimizadas a la aplicación que se está protegiendo. Para ello habría que crear una buena base de datos de vectores de ataques y peticiones legítimas con los que alimentar la aplicación de autoaprendizaje.

Ya que hay empresas que comercializan paquetes de reglas, se podría crear un sistema de actualizaciones que de forma autónoma compruebe la disponibilidad y gestione la actualización de dichos paquetes.

Por último ya que el punto de mejora más evidente es el consumo de CPU de la librería, habría que buscar puntos de mejora, hacer análisis de *performance* y optimizar su rendimiento utilizando técnicas como *zero copy* y buscando alternativas en librerías de parseo de contenido y *pipeline*.

APÉNDICES

Instalación de Libmodsecurity

Para la instalación de Libmodsecurity se han ejecutado los pasos detallados por su desarrollador:

1. Lo primero será instalar sus dependencias. La implementación de este proyecto se basa en Debian, por lo que se instalarán los paquetes desde el repositorio:

```
apt-get update && apt-get install -y automake bison build-essential \
    g++ gcc libbison-dev libcurl4-openssl-dev libfl-dev libgeoip-dev \
    liblmdb-dev libpcre3-dev libtool libxml2-dev libyajl-dev make \
    pkg-config zlib1g-dev
```

2. Una vez instaladas las dependencias se obtiene el proyecto de su repositorio GIT y se actualiza para tenerlos últimos cambios:

```
git clone --depth 1 -b v3/master \
     --single-branch https://github.com/SpiderLabs/ModSecurity

cd ModSecurity/

git submodule init

git submodule update
```

3. Por último construimos los binarios y los instalamos en una ruta del sistema

```
./build.sh && ./configure –prefix=/usr/local/modsecurity \
  && make && make install
```

Diagrama de Gantt

La primera etapa del proyecto ha consistido en analizar las soluciones disponibles, el estado del arte. En esta etapa se han montado pequeñas demos con las que familiarizarse con las soluciones y ver las capacidades que cada una de ellas ofrecía.

Una vez elegida una solución se ha comenzado con la documentación. Esta etapa ha sido siempre una tarea paralela al desarrollo, con la intención de dejar constancia de las decisiones de diseño que se iban tomando.

Una vez terminada la integración entre Pound y Libmodsecurity se ha hecho un test de rendimiento, con el que validar si el desarrollo cumplía con lo esperado.

Tras ser aprobada la integración, se comenzó con la implementación de la API. Para evitar arrastrar errores e introducir bugs en propiedades que ya habían sido testeadas, se llevó en paralelo el desarrollo de la aplicación de tests. Esta tarea ha hecho posible desarrollar de forma confiada, ya que continuamente se ha ido probando, el nuevo código y lo ya implementado hasta el momento.

Por último se han realizado pruebas con casos prácticos con las que mostrar la potencia de la solución y sus posibles usos.

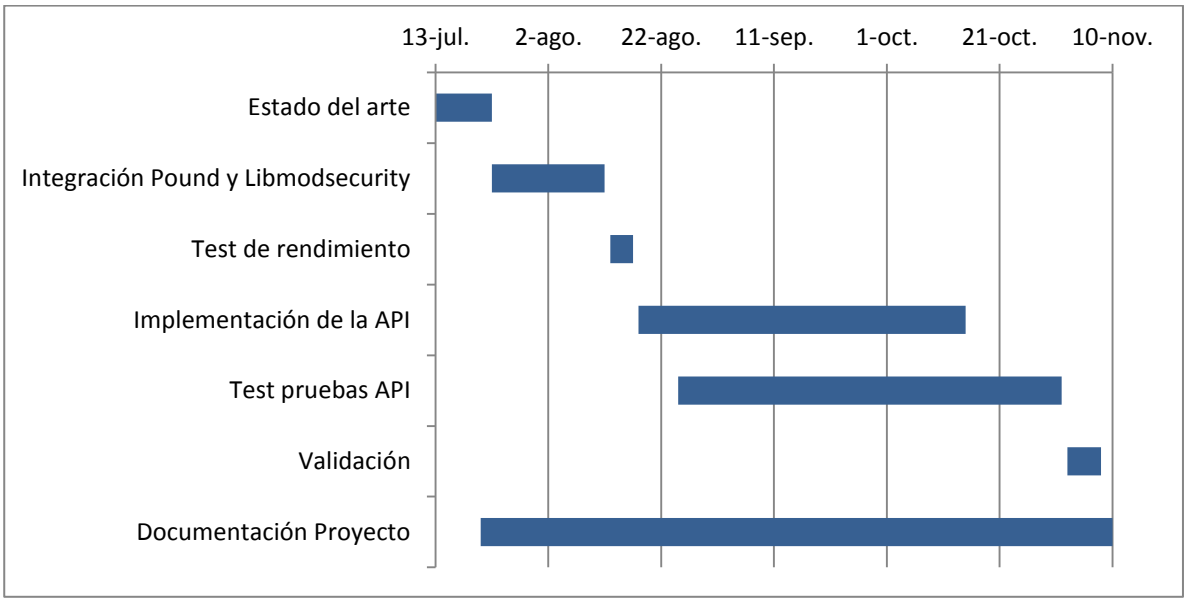

Figura 23 . Diagrama de Gannt

GLOSARIO

Backend. Es la instancia encargada del procesamiento de una petición. Comúnmente suele identificarse con el servidor.

Balanceador de carga. Es un dispositivo de red encargado de gestionar peticiones HTTP y repartirlas entre diferentes backends.

Binario. El binario es el ejecutable resultante de compilar el código fuente de un software.

Body o cuerpo HTTP. Son los datos de aplicación que circulan en una petición HTTP. Estos datos penden tras las cabeceras.

Directivas. Son sentencias SecLang que la librería Libmodsecurity aplicará sobre el tráfico que analiza. La sentencia más representativa es SecRule, que aplica una acción en función de un match.

Granja. Es una instancia de balanceo, virtualiza una interfaz de conexión (IP y puerto) para distribuir el tráfico entre un conjunto de backends.

Hash. Es uno de los tipos de datos de Perl, el hash representa una estructura formada por pares clave-valor. El valor puede ser tanto un Scalar (numérico o string) como una referencia a otra estructura. Es la forma de crear estructuras complejas.

Headers o cabeceras HTTP. Son cadenas enviadas tanto por el cliente como por el backend con las que identificar el tipo de datos que se están enviando y cómo deben ser interpretados.

Libmodsecurity. Es la librería utilizada para analizar el tráfico y buscar vulnerabilidades. Implementa las capacidades de WAF del proyecto.

Match. Es la acción de que una condición o patrón se cumpla.

Método o verbo HTTP. Es uno de los campos de una petición HTTP, indica el tipo de acción que se desea hacer.

Operador. Es uno de los campos de una directiva SecRule. Indica qué tipo de operación hay que aplicar para encontrar un match.

Parsear. Es la acción de identificar un conjunto de datos en bruto en elementos conocidos.

Path HTTP. Es la ruta al recurso sobre el que se quiere actuar en una petición HTTP.

Pool de backends. Es un conjunto de backends que sirven las mismas peticiones.

Pound. Es el software usado para desempeñar las tareas de balanceo de carga.

Poundctl. Es el binario usado para enviar instrucciones a un proceso de Pound. Se comunican mediante un socket UNIX de control.

Regla. Una regla es un objeto creado a través de la API que posteriormente queda traducido en una directiva SecLang.

Seclang. Es el lenguaje creado para las directivas de Libmodsecurity. Identifica el formato de las directivas y la coherencia de debe existir entre algunas sentencias.

Set. Es un objeto de la API encargado de organizar las reglas. Se compone de una sección de configuración y una lista ordenada de las reglas definidas.

Variable. Es uno de los campos de una directiva SecLang, en concreto el que indica en qué elemento de la trama HTTP hay que aplicar el operador.

WAF. Son las siglas de Web Application Firewall. Es una tecnología encargada de aceptar o denegar peticiones HTTP en función a unas políticas de seguridad.

BIBLIOGRAFÍA

http://www.apsis.ch/pound

https://malware.expert/tutorial/writing-modsecurity-rules/

https://www.modsecurity.org/CRS/Documentation/making.html

https://support.kemptechnologies.com/hc/en-us/articles/209635223-How-to-write-a-WAF-rule-Modsecurity-Rule-Writing

https://www.feistyduck.com/library/modsecurity-handbook-free/online/ch03-configuration.html

https://coreruleset.org/installation/

http://blog.zimmerle.org/2016/01/an-overview-of-upcoming-libmodsecurity.html

https://github.com/zevenet/zlb/

https://code-maven.com/set-up-cgi-with-apache

https://github.com/defanator/modsecurity-performance/wiki

https://www.fwhibbit.es/toolkits-para-hacking-iii-maquinas-vulnerables-para-practicar

ANEXO A: WAF.C

Este anexo se muestra el contenido del fichero *waf.c*. Es un fichero librería para Pound creado para soportar WAF. Este fichero se incluirá en el repositorio de Pound.

Waf_del_transaction

Esta función borra la transacción y deja el puntero apuntando a *null* para indicar que no existe ninguna transacción.

```
void
waf_del_transaction(Transaction **transac) {
  if (*transac != NULL) {
    msc_transaction_cleanup(*transac);
    *transac = NULL;
  }
}
```

Waf_create_transaction

Esta función es la encargada de inicializar la transacción, es llamada cuando se recibe una petición. La transacción creada debe ser eliminada antes de recibir otra petición del cliente.

```
void
waf_create_transaction(Transaction **t, ModSecurity *ms, Rules
*rules) {
  *t = msc_new_transaction( ms, rules, &logmsg);
}
```

Waf_check_rule

Comprueba la sintaxis de una regla. Para ello crea un set, y le asigna la regla, si existe un error durante el proceso, la regla se da por no válida.

```
int
waf_check_rule(char *rule_str) {
  // waf rules is a global variable
  char *err_msg = NULL;
  Rules *rule = NULL;
  int err_rul = 0;

  rule = msc_create_rules_set();
  //msc_rules_add
```

```
  msc_rules_add( rule, rule_str, &err_msg);

  if (err_msg) {
    err_rul = 1;
    logmsg (LOG_ERR,"Error loading waf rules, %s", err_msg);
    free (err_msg);
  }

  msc_rules_cleanup(rule);

  return err_rul;
}
```

Waf_check_set

Comprueba la integridad de un fichero de reglas. Para ello crea un set, y le asigna el fichero, si existe un error durante el proceso, la el fichero se da por no válido.

```
int
waf_check_set(char *file) {
  // waf rules is a global variable
  char *err_msg = NULL;
  Rules *rule = NULL;
  int err_rul = 0;

  rule = msc_create_rules_set();
  //msc_rules_add
  msc_rules_add_file( rule, file, &err_msg);

  if (err_msg) {
    err_rul = 1;
    logmsg(LOG_ERR,"Error loading waf rules, %s", err_msg);
    free (err_msg);
  }

  msc_rules_cleanup(rule);

  return err_rul;
}
```

Waf_reload_rules

Recarga el set de reglas global de la granja tomando el valor que haya en el momento en el fichero de configuración. Este set es usado por cada uno de las conexiones para crear sus propias transacciones. Si hubiera algún problema durante la recarga del set de reglas, se mantendría el set anterior.

```
void
waf_reload_rules(void) {
```

```
  // waf rules is a global variable
  char *err = NULL;

  if ( waf_rules ) {
    msc_rules_cleanup(waf_rules);
    waf_rules = NULL;
  }

  if ( waf_rules_file) {
    waf_rules = msc_create_rules_set();

    for ( FILE_LIST *it = waf_rules_file; it != NULL; it=it->next ) {
      //msc_rules_add
      msc_rules_add_file( waf_rules, it->file, &err);

      if (err) {
        waf_rules = NULL;
        logmsg(LOG_ERR,"Error loading waf rules, %s", err);
        free (err);
      }
    }
  }
}
```

Waf_body_enabled

Esta función sentencia si el body debe ser analizado en función de la configuración, el tamaño y las cabeceras que se han recibido.

```
int
waf_body_enabled(int bodybuf, const char *logtag, \
    int body_size, int chunked, int rpc) {
  int ret=0;
  if (!body_size) {
  } else if(!bodybuf) {
  } else if(body_size >= bodybuf && bodybuf >0) {
  } else if (chunked){
  } else if (rpc==1){
  } else {
    ret = 1;
  }
  return ret;
}
```

Parse_headers

Toma las cabeceras recibidas por el cliente o el servidor y monta un vector que entiende la librería Libmodsecurity. Para ello se guardan punteros de donde empieza cada una de las claves y los valores de las cabeceras, así no se utiliza memoria extra para comunicarse con la librería.

```
int
parse_headers( const char *header, char **key, \
    int *key_size,char **value, int *value_size ) {
  int fin = 0;
  int flag_clr=0;
  int parsing_value=0;
  int i;
  *key = header;
  *key_size = 0;
  *value_size = 0;

  if (header == NULL)
    return -1;

  // look for the separator, ':'
  for ( i=0; i<MAXBUF && !fin; i++ ) {
    if (!parsing_value) {
      // end key
      if (header[i] == ':')
        fin = 1;
      else if (header[i] == '\0')
        fin = -1;  //wrong parse
    }
  }

  if (fin == 1) {
    i--;
    *key_size = i;
    *value=header+i+2;
    *value_size = strlen(header)-2-*key_size; // rest size of " :
  }

  return fin;
}
```

Waf_add_http_info

Toma la primera línea de la cabecera enviada por el cliente y parsea los datos: verbo, URI y versión HTTP. Tras eso pasa la información a la librería.

```
int
waf_add_http_info(Transaction *t, const char *header) {
  int ret = 0;
  char version[5];
  char verb[20];
  char uri[512];
  int i = 0;
  int str_ind=0;
  int vers_flag= 0;
  int size = strlen(header);
  int fin;
```

```
  // parse verb
  for (fin=0,str_ind=0;!fin && i<size; i++, str_ind++) {
    if ( header[i] == ' ' ) {
      verb[str_ind] = '\0';
      fin=1;
    }
    else
      verb[str_ind] = header[i];
  }

  // parse path
  for (fin=0,str_ind=0;!fin && i<size; i++, str_ind++) {
    if ( header[i] == ' ' ) {
      uri[str_ind] = '\0';
      fin=1;
    }
    else
      uri[str_ind] = header[i];
  }

  // parse version
  for (fin=0,str_ind=0;!fin && i<size; i++) {
   if (header[i] == '\0')
     fin=1;

   else {
    if ( vers_flag ) {
     version[str_ind]=header[i];
     str ind++;
    }
    else if (!vers_flag && header[i] == '/')
      vers_flag=1;
    }
  }

  msc_process_uri(t, uri, verb, version);

  return ret;
}
```

Waf_add_req_head

Toma la petición del cliente y extrae la línea con los campos relativos al verbo, URI y versión y las cabeceras. Tras ello manda los datos a las funciones que se encargan de su procesamiento *waf_add_http_info* y *parse_headers*.

```
int
waf_add_req_head(Transaction *t,char const **headers,int num_headers)
{
  char *key;
  int key_size;
  char *value;
```

```
  int value_size;
  int ret = 1;
  int cont=1;

  waf_add_http_info(t, headers[0]);

  // skip first header, it is the VERB, URI and VERSION
  for(int i = 1; cont == 1 && i<num_headers; i++) {
     cont = parse_headers( headers[i], &key, &key_size, &value, \
       &value_size);
     if ( cont == 1 )
       msc_add_n_request_header(t, key, key_size, value, value_size);
     else {
       ret = 0;
     }
  }

  msc_process_request_headers(t);

  return ret;
}
```

Waf_add_resp_head

Toma las cabeceras de respuesta del servidor, extrae el código HTTP y manda las cabeceras a la Libmodsecurity a través de la función *parse_headers*.

```
int
waf_add_resp_head(Transaction *t, char const **headers, int \
    num_headers) {
  char *key;
  int key_size;
  char *value;
  int value_size;
  int ret = 1;
  int cont=1;
  int http_code;
  char http_code_str[4];
  char http_version[9];
  char const *p;

  // parse http response code
  // parse version
  p=headers[0];
  int size = strlen(p);
  int param=0;
  char aux;
  for (int i=0, ic=0; param<2 && i<size; i++, ic++) {
    aux=p[i];
    if ( aux == ' ' )
      aux = '\0';
```

```
    if (param == 0)
      http_version[ic]=aux;
    else if (param == 1)
      http_code_str[ic]=aux;

    if ( aux == '\0' ) {
      ic=-1;
      param++;
    }
  }
  http_code=atoi(http_code_str);

  // parse response headers
  for(int i = 1; cont == 1 && i<num_headers; i++) {
     cont = parse_headers( headers[i], &key, &key_size, &value, \
       &value_size);
     if ( cont == 1 )
       msc_add_n_response_header(t, key, key_size, value, \
        value_size);
     else {
       ret = 0;
     }
  }

  msc_process_response_headers(t, http_code, http_version);

  return ret;
}
```

Read_body

Lee el socket de cliente o servidor y toma el body. El body es guardado en un buffer para posteriormente analizarlo y reenviarlo a su destinatario original. Esta función es ejecutada tras haber obtenido todas las cabeceras.

```
int
read_body(BIO *sock, char **buff, int size)
{
  if (*buff)
    logmsg(LOG_ERR, "(%lx) body buffer is busy", pthread_self());
  else {
    *buff=(char *)malloc(size*sizeof(char));
    while(BIO_read(sock, *buff, size) < size){}
  }
}
```

Waf_resolution

Esta función comprueba el estado de Libmodsecurity, debe ejecutarse después de que ésta haya analizado

algún dato, para ver cuál ha sido su resolución.

```
int
waf_resolution(Transaction *t,int *int_code, char **url) {

  ModSecurityIntervention intervention;
  intervention.status = 200;
  intervention.url = NULL;
  intervention.log = NULL;
  intervention.disruptive = 0;
  WAF_ACTION waf_action = ALLOW;

  if (msc_intervention(t,&intervention)) {

    msc_process_logging(t);

    if (intervention.url) {
      waf_action = REDIRECTION;
      if (intervention.status == 200)
        intervention.status = 302;  // default value
    } else if (intervention.disruptive) {
      waf_action = BLOCK;
      if (intervention.status == 200)
        intervention.status = 403;  // default value
    }
  }

  return waf_action;
}
```

ANEXO B: EJEMPLOS LLAMADAS API

El contenido de este anexo es un ejemplo de petición-respuesta para cada una de las llamadas de la API. Las reglas creadas en estas pruebas no tienen validez funcional, son solo ejemplos para comprobar el correcto funcionamiento de la gestión ejercida por la API.

Creación de un set

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d '{"name":"test_waf_ori"}'
http://192.168.100.150:80/api.cgi/waf
```

<< Respuesta:

```
{
   "description" : "Create the WAF set, test_waf_ori",
   "message" : "Added the WAF set test_waf_ori",
   "params" : {
      "configuration" : {
         "audit" : "false",
         "default_action" : "allow",
         "default_log" : "",
         "default_phase" : "1",
         "disable_rules" : [],
         "process_request_body" : "false",
         "process_response_body" : "false",
         "request_body_limit" : "",
         "status" : "false"
      },
      "rules" : []
   }
}
```

Crear una regla

>> Petición:

```
curl -X -H 'Content-Type: application/json'
-d '{"log_data":"match in rule 41", \
"set_variable":["tx.php_injection_score=+%{tx.critical_anomaly_score}
","tx.anomaly_score=+%{tx.critical_anomaly_score}","tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-%{matched_var_name}=%{tx.0}"], \
"severity":2,"skip_after":100,"description":"Testing rule", \
"log":"true","transformations":["base64Decode","escapeSeqDecode","url
Decode"], \
"audit":"false","operator":"strContains","execute":"/opt/example_scri
pt.lua","action":"pass", \
"variables":["REQUEST_URI","REQUEST_BODY"],"multi_match":"true","http
_code":200, \
```

```
  "capture":"false","skip":2,"tag":["testing","Api
tests"],"rule_id":199,"operating":"index","phase":2}'
  http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules
```

<< Respuesta:

```
  {
     "description" : "Create a rule in the set test_waf_ori",
     "message" : "The rule was created properly.",
     "params" : {
        "action" : "pass",
        "audit" : "false",
        "capture" : "false",
        "chain" : [],
        "description" : "Testing rule",
        "execute" : "/opt/example_script.lua",
        "http_code" : "200",
        "id" : 0,
        "log" : "true",
        "log_data" : "match in rule 41",
        "multi_match" : "true",
        "operating" : "index",
        "operator" : "strContains",
        "phase" : "2",
         "raw" : "SecRule REQUEST_URI|REQUEST_BODY \"@contains index\"
\"id:199,msg:'Testing                          rule',tag:testing,tag:Api
tests,severity:2,phase:2,t:base64Decode,t:escapeSeqDecode,t:urlDecode,
multimatch,pass,status:200,log,noauditlog,logdata:match     in     rule
41,setvar:tx.php_injection_score=+%{tx.critical_anomaly_score},setvar:
tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:2,skipAfter:100,exec:/opt/example_scr
ipt.lua\"",
        "rule_id" : "199",
        "set_variable" : [
           "tx.php_injection_score=+%{tx.critical_anomaly_score}",
           "tx.anomaly_score=+%{tx.critical_anomaly_score}",
                     "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
        ],
        "severity" : "2",
        "skip" : "2",
        "skip_after" : "100",
        "tag" : [
           "testing",
           "Api tests"
        ],
        "transformations" : [
           "base64Decode",
           "escapeSeqDecode",
           "urlDecode"
        ],
        "type" : "match_action",
        "variables" : [
           "REQUEST_URI",
           "REQUEST_BODY"
        ]
```

```
    }
  }
```

Crear una regla

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d '{"raw":"SecRule TX:sampling_rnd100 "@rx ^0([0-9])" \
 "id:1003,phase:1,pass""}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules
```

<< Respuesta:


```
{
   "description" : "Create a rule in the set test_waf_ori",
   "message" : "The rule was created properly.",
   "params" : {
      "action" : "pass",
      "audit" : "false",
      "capture" : "false",
      "chain" : [],
      "description" : "",
      "execute" : "",
      "http_code" : "",
      "id" : 1,
      "log" : "false",
      "log_data" : "",
      "multi_match" : "false",
      "operating" : "^0([0-9])",
      "operator" : "strRegex",
      "phase" : "1",
       "raw"   :   "SecRule   TX:sampling_rnd100   \"@rx   ^0([0-9])\"
\"id:1003,phase:1,pass\"",
      "rule_id" : "1003",
      "set_variable" : [],
      "severity" : "",
      "skip" : "",
      "skip_after" : "",
      "tag" : [],
      "transformations" : [],
      "type" : "match_action",
      "variables" : [
         "TX:sampling_rnd100"
      ]
   }
}
```


Mostrar la regla 0

>> Petición:

```
curl -X GET
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0
```

<< Respuesta:


```
  {
     "description" : "Get the WAF rule 0 of the set test_waf_ori",
     "params" : {
        "action" : "pass",
        "audit" : "false",
        "capture" : "false",
        "chain" : [],
        "description" : "Testing rule",
        "execute" : "/opt/example_script.lua",
        "http_code" : "200",
        "id" : 0,
        "log" : "true",
        "log_data" : "match in rule 41",
        "multi_match" : "true",
        "operating" : "index",
        "operator" : "strContains",
        "phase" : "2",
         "raw" : "SecRule REQUEST_URI|REQUEST_BODY \"@contains index\"
\"id:199,msg:'Testing                          rule',tag:testing,tag:Api
tests,severity:2,phase:2,t:base64Decode,t:escapeSeqDecode,t:urlDecode,
multimatch,pass,status:200,log,noauditlog,logdata:match     in     rule
41,setvar:tx.php_injection_score=+%{tx.critical_anomaly_score},setvar:
tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:2,skipAfter:100,exec:/opt/example_scr
ipt.lua\"",
        "rule_id" : "199",
        "set_variable" : [
           "tx.php_injection_score=+%{tx.critical_anomaly_score}",
           "tx.anomaly_score=+%{tx.critical_anomaly_score}",
                     "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
        ],
        "severity" : "2",
        "skip" : "2",
        "skip_after" : "100",
        "tag" : [
           "testing",
           "Api tests"
        ],
        "transformations" : [
           "base64Decode",
           "escapeSeqDecode",
           "urlDecode"
        ],
        "type" : "match_action",
        "variables" : [
           "REQUEST_URI",
           "REQUEST_BODY"
        ]
     }
  }
```

Crear una regla

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d '{"copy_from":199}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules
```

Respuesta:


```
{
   "description" : "Create a rule in the set test_waf_ori",
   "message" : "The rule was created properly.",
   "params" : {
      "action" : "pass",
      "audit" : "false",
      "capture" : "false",
      "chain" : [],
      "description" : "Testing rule",
      "execute" : "/opt/example_script.lua",
      "http_code" : "200",
      "id" : 2,
      "log" : "true",
      "log_data" : "match in rule 41",
      "multi_match" : "true",
      "operating" : "index",
      "operator" : "strContains",
      "phase" : "2",
       "raw" : "SecRule REQUEST_URI|REQUEST_BODY \"@contains index\"
\"id:199,msg:'Testing rule',tag:testing,tag:Api
tests,severity:2,phase:2,t:base64Decode,t:escapeSeqDecode,t:urlDecode,
multimatch,pass,status:200,log,noauditlog,logdata:match in rule
41,setvar:tx.php_injection_score=+%{tx.critical_anomaly_score},setvar:
tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:2,skipAfter:100,exec:/opt/example_scr
ipt.lua\"",
      "rule_id" : "199",
      "set_variable" : [
         "tx.php_injection_score=+%{tx.critical_anomaly_score}",
         "tx.anomaly_score=+%{tx.critical_anomaly_score}",
                  "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
      ],
      "severity" : "2",
      "skip" : "2",
      "skip_after" : "100",
      "tag" : [
         "testing",
         "Api tests"
      ],
      "transformations" : [
         "base64Decode",
         "escapeSeqDecode",
         "urlDecode"
      ],
      "type" : "match_action",
```

```
        "variables" : [
           "REQUEST_URI",
           "REQUEST_BODY"
        ]
     }
  }
```

Modificar la regla 0

>> Petición:

```
curl -X PUT -H 'Content-Type: application/json'
-d
'{"transformations":["replaceComments","removeCommentsChar"],"operator
":"detectSQLi","description":"Rule to modify the another
one","log":"","severity":4,"skip_after":12,"log_data":"","set_variable
":["tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"],"operating":"","phase":4,"rule_id":200,"
variables":["RESPONSE_HEADERS_NAMES"],"action":"deny","multi_match":"f
alse","http_code":402,"capture":"true","skip":1,"tag":["testing
2","Api tests 2"],"execute":""}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0
```

<< Respuesta:

```
{
   "description" : "Modify the rule 0 from the set test_waf_ori",
   "params" : {
      "action" : "deny",
      "audit" : "false",
      "capture" : "true",
      "chain" : [],
      "description" : "Rule to modify the another one",
      "execute" : "",
      "http_code" : "402",
      "id" : 0,
      "log" : "false",
      "log_data" : "",
      "multi_match" : "false",
      "operating" : "",
      "operator" : "detectSQLi",
      "phase" : "4",
       "raw" : "SecRule RESPONSE_HEADERS_NAMES \"@detectSQLi \"
\"id:200,msg:'Rule to modify the another one',tag:testing 2,tag:Api
tests
2,severity:4,phase:4,t:replaceComments,t:removeCommentsChar,capture,de
ny,status:402,noauditlog,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:1,skipAfter:12\"",
      "rule_id" : "200",
      "set_variable" : [
                  "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
      ],
      "severity" : "4",
      "skip" : "1",
```

```
        "skip_after" : "12",
        "tag" : [
           "testing 2",
           "Api tests 2"
        ],
        "transformations" : [
           "replaceComments",
           "removeCommentsChar"
        ],
        "type" : "match_action",
        "variables" : [
           "RESPONSE_HEADERS_NAMES"
        ]
     }
  }
```

Modificar la regla 2

>> Petición:

```
  curl -X PUT -H 'Content-Type: application/json'
  -d      '{"raw":"SecRule      REQUEST_METHOD      "@streq      POST"
"id:9001184,phase:1,t:none,pass,nolog,noauditlog,chain"\nSecRule
REQUEST_FILENAME "@rx /file/ajax/field_asset_[a-z0-9_]+/[ua]nd/0/form-
[a-z0-9A-Z_-]+$" "chain"\nSecRule REQUEST_COOKIES:/S?SESS[a-f0-9]+/ "@
" "ctl:requestBodyAccess=Off""}'
  http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/2
```

<< Respuesta:

```
  {
     "description" : "Modify the rule 2 from the set test_waf_ori",
     "params" : {
        "action" : "pass",
        "audit" : "false",
        "capture" : "false",
        "chain" : [
           {
              "capture" : "false",
              "execute" : "",
              "multi_match" : "false",
                  "operating"     :     "/file/ajax/field_asset_[a-z0-
9_]+/[ua]nd/0/form-[a-z0-9A-Z_-]+$",
              "operator" : "rx",
              "set_variable" : [],
              "transformations" : [],
              "variables" : [
                 "REQUEST_FILENAME"
              ]
           },
           {
              "capture" : "false",
              "execute" : "",
              "multi_match" : "false",
              "operating" : "",
```

```
            "operator" : "",
            "set_variable" : [],
            "transformations" : [],
            "variables" : [
               "REQUEST_COOKIES:/S?SESS[a-f0-9]+/"
            ]
         }
      ],
      "description" : "",
      "execute" : "",
      "http_code" : "",
      "id" : 2,
      "log" : "false",
      "log_data" : "",
      "multi_match" : "false",
      "operating" : "POST",
      "operator" : "strEq",
      "phase" : "1",
        "raw"    :    "SecRule    REQUEST_METHOD    \"@streq    POST\"
\"id:9001184,phase:1,t:none,pass,nolog,noauditlog,chain\"\nSecRule
REQUEST_FILENAME          \"@rx          /file/ajax/field_asset_[a-z0-
9_]+/[ua]nd/0/form-[a-z0-9A-Z_-]+$\"                \"chain\"\nSecRule
REQUEST_COOKIES:/S?SESS[a-f0-9]+/ \"@ \" \"ctl:requestBodyAccess=Off\"
#",
      "rule_id" : "9001184",
      "set_variable" : [],
      "severity" : "",
      "skip" : "",
      "skip_after" : "",
      "tag" : [],
      "transformations" : [
         "none"
      ],
      "type" : "match_action",
      "variables" : [
         "REQUEST_METHOD"
      ]
   }
}
```


Modificar la regla 2

>> Petición:

```
curl -X PUT -H 'Content-Type: application/json'
-d      '{"raw":"SecRule      REQUEST_METHOD      "@streq      POST"
"id:9001185,phase:1,t:none,pass,nolog,noauditlog"\nSecRule
REQUEST_COOKIES:/S?SESS[a-f0-9]+/                  "@                "
"id:9001187,ctl:requestBodyAccess=Off""}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/2
```

<< Respuesta:


```
{
   "description" : "Modify the rule 2 from the set test_waf_ori",
```

```
    "params" : {
       "action" : "pass",
       "audit" : "false",
       "capture" : "false",
       "chain" : [],
       "description" : "",
       "execute" : "",
       "http_code" : "",
       "id" : 2,
       "log" : "false",
       "log_data" : "",
       "multi_match" : "false",
       "operating" : "POST",
       "operator" : "strEq",
       "phase" : "1",
          "raw"    :    "SecRule    REQUEST_METHOD    \"@streq    POST\"
\"id:9001185,phase:1,t:none,pass,nolog,noauditlog\"",
       "rule_id" : "9001185",
       "set_variable" : [],
       "severity" : "",
       "skip" : "",
       "skip_after" : "",
       "tag" : [],
       "transformations" : [
          "none"
       ],
       "type" : "match_action",
       "variables" : [
          "REQUEST_METHOD"
       ]
    }
 }
```


Mostrar el set "test_waf_ori"

>> Petición:

```
curl -X GET
http://192.168.100.150:80/api.cgi/waf/test_waf_ori
```

<< Respuesta:


```
 {
    "description" : "Get the WAF set test_waf_ori",
    "params" : {
       "configuration" : {
          "audit" : "false",
          "default_action" : "allow",
          "default_log" : "false",
          "default_phase" : "1",
          "disable_rules" : [],
          "process_request_body" : "false",
          "process_response_body" : "false",
          "request_body_limit" : "",
          "status" : "false"
```

```
        },
        "rules" : [
           {
              "action" : "deny",
              "audit" : "false",
              "capture" : "true",
              "chain" : [],
              "description" : "Rule to modify the another one",
              "execute" : "",
              "http_code" : "402",
              "id" : 0,
              "log" : "false",
              "log_data" : "",
              "multi_match" : "false",
              "operating" : "",
              "operator" : "detectSQLi",
              "phase" : "4",
               "raw" : "SecRule RESPONSE_HEADERS_NAMES \"@detectSQLi \"
\"id:200,msg:'Rule to modify the another one',tag:testing 2,tag:Api
tests
2,severity:4,phase:4,t:replaceComments,t:removeCommentsChar,capture,de
ny,status:402,noauditlog,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:1,skipAfter:12\"",
              "rule_id" : "200",
              "set_variable" : [
                     "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
              ],
              "severity" : "4",
              "skip" : "1",
              "skip_after" : "12",
              "tag" : [
                 "testing 2",
                 "Api tests 2"
              ],
              "transformations" : [
                 "replaceComments",
                 "removeCommentsChar"
              ],
              "type" : "match_action",
              "variables" : [
                 "RESPONSE_HEADERS_NAMES"
              ]
           },
           {
              "action" : "pass",
              "audit" : "false",
              "capture" : "false",
              "chain" : [],
              "description" : "",
              "execute" : "",
              "http_code" : "",
              "id" : 1,
              "log" : "false",
              "log_data" : "",
              "multi_match" : "false",
```

```
            "operating" : "^0([0-9])",
            "operator" : "strRegex",
            "phase" : "1",
             "raw" : "SecRule TX:sampling_rnd100 \"@rx ^0([0-9])\"
\"id:1003,phase:1,pass\"",
            "rule_id" : "1003",
            "set_variable" : [],
            "severity" : "",
            "skip" : "",
            "skip_after" : "",
            "tag" : [],
            "transformations" : [],
            "type" : "match_action",
            "variables" : [
               "TX:sampling_rnd100"
            ]
         },
         {
            "action" : "pass",
            "audit" : "false",
            "capture" : "false",
            "chain" : [],
            "description" : "",
            "execute" : "",
            "http_code" : "",
            "id" : 2,
            "log" : "false",
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "POST",
            "operator" : "strEq",
            "phase" : "1",
              "raw" : "SecRule REQUEST_METHOD \"@streq POST\"
\"id:9001185,phase:1,t:none,pass,nolog,noauditlog\"",
            "rule_id" : "9001185",
            "set_variable" : [],
            "severity" : "",
            "skip" : "",
            "skip_after" : "",
            "tag" : [],
            "transformations" : [
               "none"
            ],
            "type" : "match_action",
            "variables" : [
               "REQUEST_METHOD"
            ]
         },
         {
            "action" : "",
            "audit" : "false",
            "capture" : "false",
            "chain" : [],
            "description" : "",
            "execute" : "",
            "http_code" : "",
```

```
                "id" : 3,
                "log" : "false",
                "log_data" : "",
                "multi_match" : "false",
                "operating" : "",
                "operator" : "",
                "phase" : "",
                "raw" : "SecRule REQUEST_COOKIES:/S?SESS[a-f0-9]+/ \"@ \"
\"id:9001187,ctl:requestBodyAccess=Off\"",
                "rule_id" : "9001187",
                "set_variable" : [],
                "severity" : "",
                "skip" : "",
                "skip_after" : "",
                "tag" : [],
                "transformations" : [],
                "type" : "match_action",
                "variables" : [
                   "REQUEST_COOKIES:/S?SESS[a-f0-9]+/"
                ]
             }
          ]
       }
    }
```

Mover la regla 0 a la posición 2

>> Petición:

```
Curl -X POST -H 'Content-Type: application/json'
-d '{"position":2}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0/actions
```

<< Respuesta:

```
{
   "description" : "Move a rule in the set test_waf_ori",
   "message" : "The rule was moved properly to the position 2."
}
```

Mostrar el set "test_waf_ori"

>> Petición:

```
curl -X GET
http://192.168.100.150:80/api.cgi/waf/test_waf_ori
```

<< Respuesta:

```
{
   "description" : "Get the WAF set test_waf_ori",
   "params" : {
      "configuration" : {
         "audit" : "false",
```

```
            "default_action" : "allow",
            "default_log" : "false",
            "default_phase" : "1",
            "disable_rules" : [],
            "process_request_body" : "false",
            "process_response_body" : "false",
            "request_body_limit" : "",
            "status" : "false"
         },
         "rules" : [
            {
               "action" : "pass",
               "audit" : "false",
               "capture" : "false",
               "chain" : [],
               "description" : "",
               "execute" : "",
               "http_code" : "",
               "id" : 0,
               "log" : "false",
               "log_data" : "",
               "multi_match" : "false",
               "operating" : "^0([0-9])",
               "operator" : "strRegex",
               "phase" : "1",
                "raw" : "SecRule TX:sampling_rnd100 \"@rx ^0([0-9])\"
\"id:1003,phase:1,pass\"",
               "rule_id" : "1003",
               "set_variable" : [],
               "severity" : "",
               "skip" : "",
               "skip_after" : "",
               "tag" : [],
               "transformations" : [],
               "type" : "match_action",
               "variables" : [
                  "TX:sampling_rnd100"
               ]
            },
            {
               "action" : "pass",
               "audit" : "false",
               "capture" : "false",
               "chain" : [],
               "description" : "",
               "execute" : "",
               "http_code" : "",
               "id" : 1,
               "log" : "false",
               "log_data" : "",
               "multi_match" : "false",
               "operating" : "POST",
               "operator" : "strEq",
               "phase" : "1",
                 "raw" : "SecRule REQUEST_METHOD \"@streq POST\"
\"id:9001185,phase:1,t:none,pass,nolog,noauditlog\"",
```

```
            "rule_id" : "9001185",
            "set_variable" : [],
            "severity" : "",
            "skip" : "",
            "skip_after" : "",
            "tag" : [],
            "transformations" : [
               "none"
            ],
            "type" : "match_action",
            "variables" : [
               "REQUEST_METHOD"
            ]
         },
         {
            "action" : "deny",
            "audit" : "false",
            "capture" : "true",
            "chain" : [],
            "description" : "Rule to modify the another one",
            "execute" : "",
            "http_code" : "402",
            "id" : 2,
            "log" : "false",
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "detectSQLi",
            "phase" : "4",
             "raw" : "SecRule RESPONSE_HEADERS_NAMES \"@detectSQLi \"
\"id:200,msg:'Rule to modify the another one',tag:testing 2,tag:Api
tests
2,severity:4,phase:4,t:replaceComments,t:removeCommentsChar,capture,de
ny,status:402,noauditlog,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:1,skipAfter:12\"",
            "rule_id" : "200",
            "set_variable" : [
                   "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
            ],
            "severity" : "4",
            "skip" : "1",
            "skip_after" : "12",
            "tag" : [
               "testing 2",
               "Api tests 2"
            ],
            "transformations" : [
               "replaceComments",
               "removeCommentsChar"
            ],
            "type" : "match_action",
            "variables" : [
               "RESPONSE_HEADERS_NAMES"
            ]
         },
```

```
            {
               "action" : "",
               "audit" : "false",
               "capture" : "false",
               "chain" : [],
               "description" : "",
               "execute" : "",
               "http_code" : "",
               "id" : 3,
               "log" : "false",
               "log_data" : "",
               "multi_match" : "false",
               "operating" : "",
               "operator" : "",
               "phase" : "",
               "raw" : "SecRule REQUEST_COOKIES:/S?SESS[a-f0-9]+/ \"@ \"
\"id:9001187,ctl:requestBodyAccess=Off\"",
               "rule_id" : "9001187",
               "set_variable" : [],
               "severity" : "",
               "skip" : "",
               "skip_after" : "",
               "tag" : [],
               "transformations" : [],
               "type" : "match_action",
               "variables" : [
                  "REQUEST_COOKIES:/S?SESS[a-f0-9]+/"
               ]
            }
         ]
      }
   }
```


Borrar la regla 0

>> Petición:

```
curl -X DELETE
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0
```

<< Respuesta:


```
{
   "description" : "Delete the rule 0 from the set test_waf_ori",
   "message" : "The rule 0 has been deleted properly"
}
```


Mostrar el set "test_waf_ori"

>> Petición:

```
curl -X GET
http://192.168.100.150:80/api.cgi/waf/test_waf_ori
```

<< Respuesta:


```
  {
     "description" : "Get the WAF set test_waf_ori",
     "params" : {
        "configuration" : {
           "audit" : "false",
           "default_action" : "allow",
           "default_log" : "false",
           "default_phase" : "1",
           "disable_rules" : [],
           "process_request_body" : "false",
           "process_response_body" : "false",
           "request_body_limit" : "",
           "status" : "false"
        },
        "rules" : [
           {
              "action" : "pass",
              "audit" : "false",
              "capture" : "false",
              "chain" : [],
              "description" : "",
              "execute" : "",
              "http_code" : "",
              "id" : 0,
              "log" : "false",
              "log_data" : "",
              "multi_match" : "false",
              "operating" : "POST",
              "operator" : "strEq",
              "phase" : "1",
                "raw"   :   "SecRule   REQUEST_METHOD   \"@streq   POST\"
\"id:9001185,phase:1,t:none,pass,nolog,noauditlog\"",
              "rule_id" : "9001185",
              "set_variable" : [],
              "severity" : "",
              "skip" : "",
              "skip_after" : "",
              "tag" : [],
              "transformations" : [
                 "none"
              ],
              "type" : "match_action",
              "variables" : [
                 "REQUEST_METHOD"
              ]
           },
           {
              "action" : "deny",
              "audit" : "false",
              "capture" : "true",
              "chain" : [],
              "description" : "Rule to modify the another one",
              "execute" : "",
              "http_code" : "402",
              "id" : 1,
              "log" : "false",
```

```
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "detectSQLi",
            "phase" : "4",
            "raw" : "SecRule RESPONSE_HEADERS_NAMES \"@detectSQLi \"
\"id:200,msg:'Rule to modify the another one',tag:testing 2,tag:Api
tests
2,severity:4,phase:4,t:replaceComments,t:removeCommentsChar,capture,de
ny,status:402,noauditlog,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:1,skipAfter:12\"",
            "rule_id" : "200",
            "set_variable" : [
                  "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
            ],
            "severity" : "4",
            "skip" : "1",
            "skip_after" : "12",
            "tag" : [
               "testing 2",
               "Api tests 2"
            ],
            "transformations" : [
               "replaceComments",
               "removeCommentsChar"
            ],
            "type" : "match_action",
            "variables" : [
               "RESPONSE_HEADERS_NAMES"
            ]
         },
         {
            "action" : "",
            "audit" : "false",
            "capture" : "false",
            "chain" : [],
            "description" : "",
            "execute" : "",
            "http_code" : "",
            "id" : 2,
            "log" : "false",
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "",
            "phase" : "",
            "raw" : "SecRule REQUEST_COOKIES:/S?SESS[a-f0-9]+/ \"@ \"
\"id:9001187,ctl:requestBodyAccess=Off\"",
            "rule_id" : "9001187",
            "set_variable" : [],
            "severity" : "",
            "skip" : "",
            "skip_after" : "",
            "tag" : [],
```

```
            "transformations" : [],
            "type" : "match_action",
            "variables" : [
               "REQUEST_COOKIES:/S?SESS[a-f0-9]+/"
            ]
         }
      ]
   }
}
```

Crear una chain en la regla 0

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d
'{"operating":"","transformations":["replaceComments","removeCommentsC
har"],"audit":"","operator":"!detectSQLi","log":"","variables":["!RESP
ONSE_BODY"],"multi_match":"false","capture":"true","execute":"","set_v
ariable":["tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"]}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0/chains
```

<< Respuesta:

```
{
   "description" : "Create a chain in the rule 0 for the set
test_waf_ori",
   "message" : "Settings were changed successful.",
   "params" : {
      "action" : "pass",
      "audit" : "false",
      "capture" : "false",
      "chain" : [
         {
            "capture" : "true",
            "execute" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "!detectSQLi",
            "set_variable" : [
                  "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
            ],
            "transformations" : [
               "replaceComments",
               "removeCommentsChar"
            ],
            "variables" : [
               "!RESPONSE_BODY"
            ]
         }
      ],
      "description" : "",
      "execute" : "",
```

```
        "http_code" : "",
        "id" : 0,
        "log" : "false",
        "log_data" : "",
        "multi_match" : "false",
        "operating" : "POST",
        "operator" : "strEq",
        "phase" : "1",
           "raw"    :    "SecRule    REQUEST_METHOD    \"@streq    POST\"
\"id:9001185,phase:1,t:none,pass,nolog,noauditlog,chain\"\nSecRule
!RESPONSE_BODY                    \"!@detectSQLi                    \"
\"t:replaceComments,t:removeCommentsChar,capture,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-%{matched_var_name}=%{tx.0}\" #",
        "rule_id" : "9001185",
        "set_variable" : [],
        "severity" : "",
        "skip" : "",
        "skip_after" : "",
        "tag" : [],
        "transformations" : [
           "none"
        ],
        "type" : "match_action",
        "variables" : [
           "REQUEST_METHOD"
        ]
     }
  }
```


Modificar la cadena 0 de la regla 0

>> Petición:


```
  curl -X PUT -H 'Content-Type: application/json'
  -d
'{"capture":"true","multi_match":"false","variables":["&RESPONSE_BODY"
],"set_variable":["tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"],"execute":"","operator":"!detectSQLi","t
ransformations":["replaceComments","removeCommentsChar"],"operating":"
","log":""}'
  http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0/chains/0
```


<< Respuesta:


```
  {
     "description"  :  "Modify  the  chain  0  in  rule  0  for  the  set
test_waf_ori",
     "params" : {
        "action" : "pass",
        "audit" : "false",
        "capture" : "false",
        "chain" : [
           {
              "capture" : "true",
              "execute" : "",
```

```
                "multi_match" : "false",
                "operating" : "",
                "operator" : "!detectSQLi",
                "set_variable" : [
                        "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
                ],
                "transformations" : [
                   "replaceComments",
                   "removeCommentsChar"
                ],
                "variables" : [
                   "!RESPONSE_BODY"
                ]
             }
          ],
          "description" : "",
          "execute" : "",
          "http_code" : "",
          "id" : 0,
          "log" : "false",
          "log_data" : "",
          "multi_match" : "false",
          "operating" : "POST",
          "operator" : "strEq",
          "phase" : "1",
            "raw"    :    "SecRule    REQUEST_METHOD    \"@streq    POST\"
\"id:9001185,phase:1,t:none,pass,nolog,noauditlog,chain\"\nSecRule
!RESPONSE_BODY                    \"!@detectSQLi                    \"
\"t:replaceComments,t:removeCommentsChar,capture,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-%{matched_var_name}=%{tx.0}\" #",
          "rule_id" : "9001185",
          "set_variable" : [],
          "severity" : "",
          "skip" : "",
          "skip_after" : "",
          "tag" : [],
          "transformations" : [
             "none"
          ],
          "type" : "match_action",
          "variables" : [
             "REQUEST_METHOD"
          ]
       }
    }
```


Borrar la cadena 0 de la regla 0

>> Petición:

```
curl -X DELETE
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0/chains/0
```

<< Respuesta:

```
  {
     "description" : "Delete the chain 0 from rule 0 for the set
test_waf_ori",
     "message" : "The chain 0 has been deleted properly"
  }
```

Crear una regla

>> Petición:

```
  curl -X POST -H 'Content-Type: application/json'
  -d '{"mark":"mark_testing"}'
  http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules
```

<< Respuesta:

```
  {
     "description" : "Create a rule in the set test_waf_ori",
     "message" : "The rule was created properly.",
     "params" : {
        "id" : 3,
        "mark" : "mark_testing",
        "raw" : "SecMarker mark_testing",
        "type" : "marker"
     }
  }
```

Modificar la regla 0

>> Petición:

```
  curl -X PUT -H 'Content-Type: application/json'
  -d '{"mark":"mark_testing"}'
  http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0
```

<< Respuesta:

```
  {
     "description" : "Modify the rule 0 from the set test_waf_ori",
     "params" : {
        "id" : 0,
        "mark" : "mark_testing",
        "raw" : "SecMarker mark_testing",
        "type" : "marker"
     }
  }
```

Crear una regla

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d
'{"skip_after":100,"severity":2,"set_variable":["tx.php_injection_scor
e=+%{tx.critical_anomaly_score}","tx.anomaly_score=+%{tx.critical_anom
aly_score}","tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"],"log_data":"match in rule
41","audit":"false","transformations":["base64Decode","escapeSeqDecode
","urlDecode"],"log":"true","description":"Testing
rule","tag":["testing","Api
tests"],"skip":2,"capture":"false","multi_match":"true","http_code":20
0,"action":"pass","execute":"/opt/example_script.lua","phase":2,"rule_
id":49}'
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules
```

<< Respuesta:

```
{
   "description" : "Create a rule in the set test_waf_ori",
   "message" : "The rule was created properly.",
   "params" : {
      "action" : "pass",
      "audit" : "false",
      "capture" : "false",
      "chain" : [],
      "description" : "Testing rule",
      "execute" : "/opt/example_script.lua",
      "http_code" : "200",
      "id" : 4,
      "log" : "true",
      "log_data" : "match in rule 41",
      "multi_match" : "true",
      "phase" : "2",
            "raw" : "SecAction \"id:49,msg:'Testing
rule',tag:testing,tag:Api
tests,severity:2,phase:2,t:base64Decode,t:escapeSeqDecode,t:urlDecode,
multimatch,pass,status:200,log,noauditlog,logdata:match in rule
41,setvar:tx.php_injection_score=+%{tx.critical_anomaly_score},setvar:
tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:2,skipAfter:100,exec:/opt/example_scr
ipt.lua\"",
      "rule_id" : "49",
      "set_variable" : [
         "tx.php_injection_score=+%{tx.critical_anomaly_score}",
         "tx.anomaly_score=+%{tx.critical_anomaly_score}",
                  "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
      ],
      "severity" : "2",
      "skip" : "2",
      "skip_after" : "100",
      "tag" : [
         "testing",
         "Api tests"
      ],
      "transformations" : [
```

```
            "base64Decode",
            "escapeSeqDecode",
            "urlDecode"
         ],
         "type" : "action"
      }
   }
```

Borrar la regla 0

>> Petición:

```
curl -X DELETE
http://192.168.100.150:80/api.cgi/waf/test_waf_ori/rules/0
```

<< Respuesta:

```
{
   "description" : "Delete the rule 0 from the set test_waf_ori",
   "message" : "The rule 0 has been deleted properly"
}
```

Copiando un set

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d '{"name":"test_waf_copied","copy_from":"test_waf_ori"}'
http://192.168.100.150:80/api.cgi/waf
```

<< Respuesta:

```
{
   "description" : "Create the WAF set, test_waf_copied",
   "message" : "Added the WAF set test_waf_copied",
   "params" : {
      "configuration" : {
         "audit" : "false",
         "default_action" : "allow",
         "default_log" : "false",
         "default_phase" : "1",
         "disable_rules" : [],
         "process_request_body" : "false",
         "process_response_body" : "false",
         "request_body_limit" : "",
         "status" : "false"
      },
      "rules" : [
         {
            "action" : "deny",
            "audit" : "false",
            "capture" : "true",
            "chain" : [],
            "description" : "Rule to modify the another one",
```

```
            "execute" : "",
            "http_code" : "402",
            "id" : 0,
            "log" : "false",
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "detectSQLi",
            "phase" : "4",
            "raw" : "SecRule RESPONSE_HEADERS_NAMES \"@detectSQLi \"
\"id:200,msg:'Rule to modify the another one',tag:testing 2,tag:Api
tests
2,severity:4,phase:4,t:replaceComments,t:removeCommentsChar,capture,de
ny,status:402,noauditlog,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:1,skipAfter:12\"",
            "rule_id" : "200",
            "set_variable" : [
                    "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
            ],
            "severity" : "4",
            "skip" : "1",
            "skip_after" : "12",
            "tag" : [
               "testing 2",
               "Api tests 2"
            ],
            "transformations" : [
               "replaceComments",
               "removeCommentsChar"
            ],
            "type" : "match_action",
            "variables" : [
               "RESPONSE_HEADERS_NAMES"
            ]
         },
         {
            "action" : "",
            "audit" : "false",
            "capture" : "false",
            "chain" : [],
            "description" : "",
            "execute" : "",
            "http_code" : "",
            "id" : 1,
            "log" : "false",
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "",
            "phase" : "",
            "raw" : "SecRule REQUEST_COOKIES:/S?SESS[a-f0-9]+/ \"@ \"
\"id:9001187,ctl:requestBodyAccess=Off\"",
            "rule_id" : "9001187",
            "set_variable" : [],
            "severity" : "",
```

```
            "skip" : "",
            "skip_after" : "",
            "tag" : [],
            "transformations" : [],
            "type" : "match_action",
            "variables" : [
               "REQUEST_COOKIES:/S?SESS[a-f0-9]+/"
            ]
         },
         {
            "id" : 2,
            "mark" : "mark_testing",
            "raw" : "SecMarker mark_testing",
            "type" : "marker"
         },
         {
            "action" : "pass",
            "audit" : "false",
            "capture" : "false",
            "chain" : [],
            "description" : "Testing rule",
            "execute" : "/opt/example_script.lua",
            "http_code" : "200",
            "id" : 3,
            "log" : "true",
            "log_data" : "match in rule 41",
            "multi_match" : "true",
            "phase" : "2",
                "raw"     :     "SecAction     \"id:49,msg:'Testing
rule',tag:testing,tag:Api
tests,severity:2,phase:2,t:base64Decode,t:escapeSeqDecode,t:urlDecode,
multimatch,pass,status:200,log,noauditlog,logdata:match     in     rule
41,setvar:tx.php_injection_score=+%{tx.critical_anomaly_score},setvar:
tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:2,skipAfter:100,exec:/opt/example_scr
ipt.lua\"",
            "rule_id" : "49",
            "set_variable" : [
               "tx.php_injection_score=+%{tx.critical_anomaly_score}"
,
               "tx.anomaly_score=+%{tx.critical_anomaly_score}",
                   "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
            ],
            "severity" : "2",
            "skip" : "2",
            "skip_after" : "100",
            "tag" : [
               "testing",
               "Api tests"
            ],
            "transformations" : [
               "base64Decode",
               "escapeSeqDecode",
               "urlDecode"
```

```
            ],
            "type" : "action"
         }
      ]
   }
}
```

Modificar un set

>> Petición:

```
curl -X PUT -H 'Content-Type: application/json'
-d
'{"status":"detection","process_response_body":"true","audit":"true","
disable_rules":[100],"request_body_limit":6456456,"process_request_bod
y":"true"}'
http://192.168.100.150:80/api.cgi/waf/test_waf_copied
```

<< Respuesta:

```
{
   "description" : "Modify the WAF set test_waf_copied",
   "params" : {
      "configuration" : {
         "audit" : "true",
         "default_action" : "allow",
         "default_log" : "false",
         "default_phase" : "1",
         "disable_rules" : [
            "100"
         ],
         "process_request_body" : "true",
         "process_response_body" : "true",
         "request_body_limit" : "6456456",
         "status" : "detection"
      },
      "rules" : [
         {
            "action" : "deny",
            "audit" : "false",
            "capture" : "true",
            "chain" : [],
            "description" : "Rule to modify the another one",
            "execute" : "",
            "http_code" : "402",
            "id" : 0,
            "log" : "false",
            "log_data" : "",
            "multi_match" : "false",
            "operating" : "",
            "operator" : "detectSQLi",
            "phase" : "4",
            "raw" : "SecRule RESPONSE_HEADERS_NAMES \"@detectSQLi \"
\"id:200,msg:'Rule to modify the another one',tag:testing 2,tag:Api
```

```
tests
2,severity:4,phase:4,t:replaceComments,t:removeCommentsChar,capture,de
ny,status:402,noauditlog,setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:1,skipAfter:12\"",
              "rule_id" : "200",
              "set_variable" : [
                     "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
              ],
              "severity" : "4",
              "skip" : "1",
              "skip_after" : "12",
              "tag" : [
                 "testing 2",
                 "Api tests 2"
              ],
              "transformations" : [
                 "replaceComments",
                 "removeCommentsChar"
              ],
              "type" : "match_action",
              "variables" : [
                 "RESPONSE_HEADERS_NAMES"
              ]
           },
           {
              "action" : "",
              "audit" : "false",
              "capture" : "false",
              "chain" : [],
              "description" : "",
              "execute" : "",
              "http_code" : "",
              "id" : 1,
              "log" : "false",
              "log_data" : "",
              "multi_match" : "false",
              "operating" : "",
              "operator" : "",
              "phase" : "",
              "raw" : "SecRule REQUEST_COOKIES:/S?SESS[a-f0-9]+/ \"@ \"
\"id:9001187,ctl:requestBodyAccess=Off\"",
              "rule_id" : "9001187",
              "set_variable" : [],
              "severity" : "",
              "skip" : "",
              "skip_after" : "",
              "tag" : [],
              "transformations" : [],
              "type" : "match_action",
              "variables" : [
                 "REQUEST_COOKIES:/S?SESS[a-f0-9]+/"
              ]
           },
           {
```

```
            "id" : 2,
            "mark" : "mark_testing",
            "raw" : "SecMarker mark_testing",
            "type" : "marker"
         },
         {
            "action" : "pass",
            "audit" : "false",
            "capture" : "false",
            "chain" : [],
            "description" : "Testing rule",
            "execute" : "/opt/example_script.lua",
            "http_code" : "200",
            "id" : 3,
            "log" : "true",
            "log_data" : "match in rule 41",
            "multi_match" : "true",
            "phase" : "2",
                "raw"     :     "SecAction     \"id:49,msg:'Testing
rule',tag:testing,tag:Api
tests,severity:2,phase:2,t:base64Decode,t:escapeSeqDecode,t:urlDecode,
multimatch,pass,status:200,log,noauditlog,logdata:match     in     rule
41,setvar:tx.php_injection_score=+%{tx.critical_anomaly_score},setvar:
tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0},skip:2,skipAfter:100,exec:/opt/example_scr
ipt.lua\"",
            "rule_id" : "49",
            "set_variable" : [
               "tx.php_injection_score=+%{tx.critical_anomaly_score}"
,
               "tx.anomaly_score=+%{tx.critical_anomaly_score}",
                   "tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/PHP_INJECTION-
%{matched_var_name}=%{tx.0}"
            ],
            "severity" : "2",
            "skip" : "2",
            "skip_after" : "100",
            "tag" : [
               "testing",
               "Api tests"
            ],
            "transformations" : [
               "base64Decode",
               "escapeSeqDecode",
               "urlDecode"
            ],
            "type" : "action"
         }
      ]
   }
}
```


Listar los sets

>> Petición:

```
curl -X GET
http://192.168.100.150:80/api.cgi/waf
```

<< Respuesta:

```
{
   "description" : "List the WAF sets",
   "params" : [
      "WORDPRESS-EXCLUSION",
      "REQUEST-903.9002-WORDPRESS-EXCLUSION-RULES",
      "REQUEST-911-METHOD-ENFORCEMENT",
      "REQUEST-901-INITIALIZATION",
      "tests",
      "RESPONSE-951-DATA-LEAKAGES-SQL",
      "owasp",
      "testWAF",
      "rule1",
      "test_waf_copied",
      "CUSTOM-RULESET",
      "test_waf_ori",
      "REQUEST-903.9001-DRUPAL-EXCLUSION-RULES"
   ]
}
```

Añadiendo el set "test_waf_ori" a una granja

>> Petición:

```
curl -X POST -H 'Content-Type: application/json'
-d '{"name":"test_waf_ori"}'
http://192.168.100.150:80/api.cgi/farms/httpTest/waf
```

<< Respuesta:

```
{
   "description" : "Apply a WAF set to a farm",
   "message" : "WAF set test_waf_ori was applied properly to the farm
httpTest.",
   "success" : "true"
}
```

Añadiendo el set "test_waf_copied" a una granja

>> Petición:

```
curl  -X POST -H 'Content-Type: application/json'
-d '{"name":"test_waf_copied"}'
http://192.168.100.150:80/api.cgi/farms/httpTest/waf
```

<< Respuesta:

```
  {
     "description" : "Apply a WAF set to a farm",
     "message" : "WAF set test_waf_copied was applied properly to the
farm httpTest.",
     "success" : "true"
  }
```

Moviendo los sets de la granja

>> Petición:

```
  curl -X POST -H 'Content-Type: application/json'
  -d '{"position":0}'
  http://192.168.100.150:80/api.cgi/farms/httpTest/waf/test_waf_copied/
actions
```

<< Respuesta:

```
  {
     "description" : "Move a set in farm",
     "message" : "The set was moved properly to the position 0."
  }
```

Quitando el set "test_waf_copied" a una granja

>> Petición:

```
  curl -X DELETE
  http://192.168.100.150:80/api.cgi/farms/httpTest/waf/test_waf_copied
```

<< Respuesta:

```
  {
     "description" : "Unset a WAF set from a farm",
     "message" : "The WAF set test_waf_copied was removed successful
from the farm httpTest.",
     "success" : "true"
  }
```

Borrando el set "test_waf_copied"

>> Petición:

```
  curl -X DELETE
  https://192.168.100.150:80/api.cgi/waf/test_waf_copied
```

<< Respuesta:

```
  {
     "description" : "Delete the WAF set test_waf_copied",
     "message" : "The WAF set test_waf_copied has been deleted
successful.",
```

```
    "success" : "true"
  }
```

Borrando el set "test_waf_ori"

>> Petición:

```
  Curl -X DELETE
  https://192.168.100.150:80/api.cgi/waf/test_waf_ori
```

<< Respuesta:

```
  {
    "description" : "Delete the WAF set test_waf_ori",
    "message" : "The WAF set test_waf_ori has been deleted
successful.",
    "success" : "true"
  }
```