UNIVERSIDAD DE BUENOS AIRE

Facultad de Ciencias Económicas, Cs. Exactas y Naturales e Ingeniería

CARRERA DE ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA

Trabajo Final de Especialización

Presentación de soluciones WAF Open Source

AUTOR: FERNANDO JAVIER CUIÑA

TUTOR: JUAN ALEJANDRO DEVINCENZI

SEPTIEMBRE DE 2021
COHORTE 2020

Índice

Resumen

Actualmente internet es una fuente de información y recursos ilimitada, pero también cuenta con una inmensa cantidad de vulnerabilidades. Existen miles de empresas que ofrecen sus servicios a través de internet, las sucursales en línea de los bancos, las financieras, las tiendas que ahora ofrecen compras en línea, sitios web del gobierno, entre muchas otras plataformas web que poseen las pequeñas, medianas y grandes empresas. Todas ellas se encuentran disponibles para sus clientes y por lo tanto, para los atacantes también. Ataques ampliamente conocidos como *Injection*, XSS (*Cross-Site Scripting*), *Broken Authentication* y otros derivados a partir de la entrada de cadenas de textos especiales, están dirigidos a vulnerabilidades en las propias aplicaciones web y no a nivel de la infraestructura de red, donde un *firewall* común y un IDS/IPS serían totalmente incapaces de proteger estos sitios contra los diferentes ataques.
El objetivo de este trabajo final es presentar tres soluciones *WAF open source* que puedan ser utilizadas en empresas para proteger sus aplicaciones web ante ataques tanto externos como internos y analizar los factores críticos que permitirán implementarlos en cualquier tipo de empresa. Se realizará un estudio de tipo explicativo y descriptivo: se analizarán las características principales de los *WAF open source* y sus funcionalidades.

La implementación, gestión y soporte de un *WAF Open Source* se presenta como una propuesta de mejora factible para cualquier organización y solucionaría muchos de los problemas de seguridad que afectan a sus aplicaciones web.

Introducción

Los desarrolladores de sitios web, aplicaciones web y las aplicaciones móviles normalmente poseen poca o nada de experiencia en cuanto a seguridad informática se refiere a la hora de su diseño y programación, las aplicaciones web son de las más atacadas para intentar conseguir acceso a la red de datos de las instituciones, esto debido a que los sitios web se encuentran expuestos al exterior.

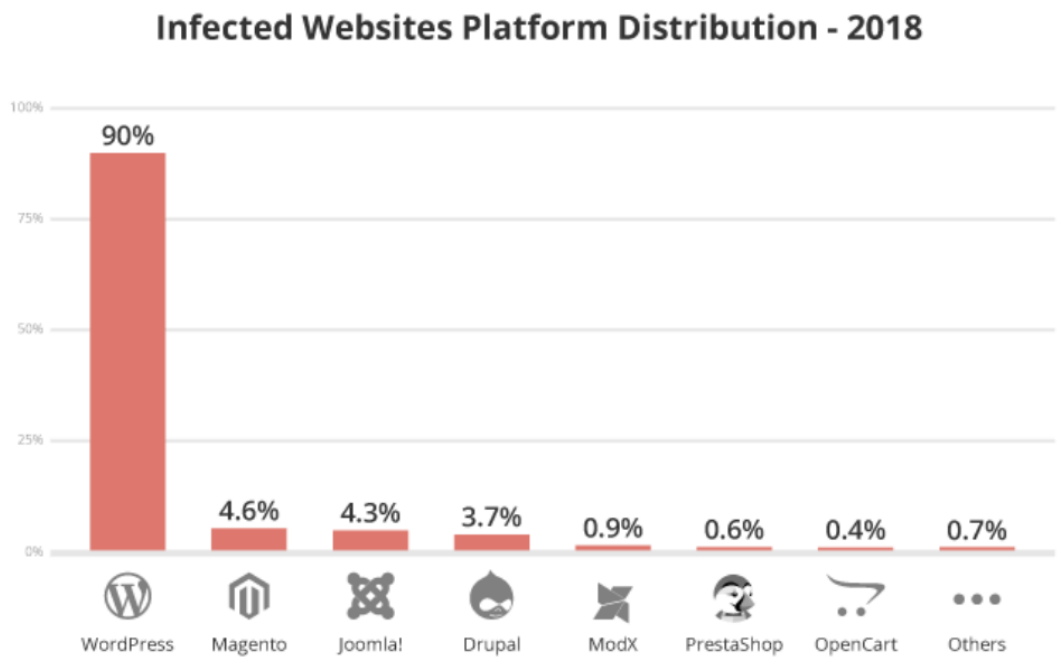


Figura 1 - Sitios web hackeados (2018). Distribución de plataformas de sitios web hackeados 2018.

Recuperado de *https://blog.sucuri.net/2019/03/hacked-website-trend-report-2018.html*

**OWASP – *Best Practice* (2008) define "*Web Application Firewall* (WAF) como una solución de seguridad en el nivel de aplicación web que, desde un punto de vista técnico, no depende de la aplicación en sí." [1]

Es uno de los métodos más usados para proteger las aplicaciones web, los cuales se interponen entre la aplicación y el atacante que intenta ingresar a la misma, explorando el tráfico entrante en la capa de aplicación del modelo de referencia OSI, con el fin de buscar patrones comunes de ataques y denegar el paso de tráfico malicioso a la aplicación web. Esta solución de software o hardware busca proteger los sitios web de las amenazas o ataques a los que puede estar expuesto.

Todas las empresas u organizaciones deben tener en cuenta la importancia de la seguridad de sus aplicaciones web, no solo para su buen funcionamiento y disponibilidad, sino también, para la continuidad del negocio.

Uno de los aspectos sumamente importantes a tener en cuenta es la cantidad de aplicaciones que poseen y que son parte de su área productiva.

El presente estudio propone presentar tres soluciones que puedan ser evaluadas por cualquier empresa u organización. Estas luego, podrán seleccionar la herramienta que mejor se adecue para su utilización.

Definición de la Problemática

La seguridad de la información de las empresas es un tema que cada vez cobra mayor fuerza en el mundo de los negocios, ya que las empresas dependen mayoritariamente del flujo de información, si esta información se distorsiona (por intrusiones no autorizadas) puede afectar significativamente la reputación de la empresa y el resultado del negocio.

Existen personas mal intencionadas (tanto fuera, como dentro de las empresas) que intentan acceder de forma no autorizada a los datos sensibles de las empresas, dicho acceso no autorizado a una red informática puede ocasionar en su mayoría problemas graves como perdida de información, suplantación de identidad, fraude, falsificaciones, etc. Las empresas sufren incidentes que podrían haberse evitado si los mecanismos de protección hubieran sido reforzados adecuadamente en su momento.

Los sistemas o aplicaciones web están expuestos a un sinnúmero de amenazas y por lo tanto se debe buscar la manera de mitigarlas. Los Sistemas de Detección de Intrusos (IDS) o los Sistemas de Prevención de Intrusos (IPS), son una solución externa a la aplicación web que no implica la modificación de su código fuente para tratar de protegerla. Esos IDS/IPS se implementan en la infraestructura de red y sirven para el monitoreo de eventos y para buscar comportamientos anómalos o amenazas en la red que puedan llegar a comprometer la integridad de los sistemas de información.

Los IDS se han aplicado de una forma general, analizando el tráfico de distintos protocolos, tales como TCP, FTP o HTTP. Los WAF se puede decir que son un caso especial de los IDS, ya que se especializan en analizar exclusivamente el tráfico HTTP/HTTPS, con el objetivo de resguardas los sitios o aplicaciones web.

El incremento exponencial de los datos que circulan en la red y la creciente sofisticación de las herramientas de ataques, hacen necesario que las

empresas cuenten con un mecanismo de protección que sea efectivo y eficiente.

Es común que las organizaciones como parte de su análisis interno indaguen sobre la siguiente interrogante ¿el sitio web de la organización se encuentra preparado para cualquier ataque de hackers? Creo que la respuesta sería ¡No!, nadie lo está. Es por esta razón que en este trabajo se presentan tres soluciones *WAF Open Source*, cualquiera de ellas ayudaría a mitigar los ataques informáticos a los sitios web, evitaría que uno de los activos más importante en la organización como lo son los datos, sean robados, alterados o eliminados.

Definición de los Objetivos

Objetivo General

Presentar tres soluciones *WAF Open Source* y determinar las ventajas y desventajas de cada una de ellas, para que cualquier empresa pueda decidir cuál de ellas es más conveniente para su implementación dentro de la misma.

Objetivos Específicos

✓ Conocer los mecanismos de ataque que son utilizados a la hora de intentar vulnerar los sitios web para tomar medidas de protección.

✓ Presentar cada una de las aplicaciones *Open Sourse* elegidas.

✓ Presentar los principales usos de cada una de ellas.

Marco Teórico

1) ¿Qué es un *Web Application Firewall*?

En el presente trabajo se abordará al WAF como una solución de seguridad en la capa siete del nivel de referencia OSI (*Open Systems Interconnection*).

El trabajo se centra en la presentación de las funciones de seguridad proporcionados por un *WAF Open Source*.

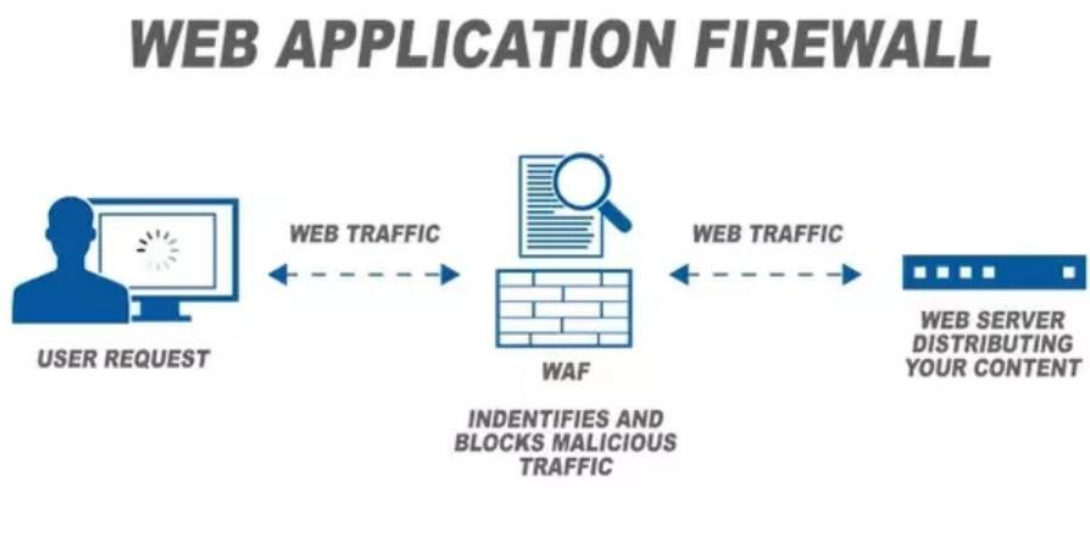


Figura 2 – *Web Application Firewall*.
https://blogs.ugr.es/seguridadinformatica/protege-tus-activos-web-con-waf-de-codigo-abierto-open-source-waf/

Los aspectos de despliegue dentro de la infraestructura de red de las organizaciones, ya sea como un dispositivo de hardware o un complemento de software para un servidor web, no son parte de este trabajo.

Cuando se inicia un proyecto de realización de un sistema web, todo programador y mánager del proyecto parte de la premisa de que cada aplicación web debe desarrollarse lo más segura posible, siguiendo alguna metodología o buenas prácticas de programación. Pero las vulnerabilidades de los sitios web son detectadas posteriormente en el ciclo de vida y es en este ciclo donde el riesgo de un ataque informático exitoso es mayor. Siendo los WAF una herramienta para diversas medidas de seguridad, nos enfocaremos solo en la protección de los sistemas web.

En la fase de desarrollo, a la hora de realizar el análisis del código fuente, ayudan a detectar y a su vez corregir las vulnerabilidades de los sistemas. A su vez la realización de pruebas de penetración que son llevadas a cabo por expertos nos da una idea de las vulnerabilidades encontradas en la aplicación web analizada y cuantas de ellas son mitigadas por el WAF.

Es en todo este contexto que la función principal de un WAF es asegurar las aplicaciones web contra vulnerabilidades detectadas y con el menor esfuerzo posible, de manera tal que no puedan ser vulneradas por atacantes. Todo esto se convierte en una tarea muy difícil por el alto grado de complejidad de la no tan típica infraestructura de aplicaciones web: servidores web, servidores de aplicaciones, framework, etc.

El objetivo principal de usar un WAF es asegurar las aplicaciones web existentes, donde los cambios requeridos dentro de la aplicación ya no pueden ser implementados a corto plazo. Esto aplica a las vulnerabilidades encontradas a través de las pruebas de penetración y/o a través del análisis del código fuente.

Los análisis de Test de Penetración sirven también para determinar el nivel de seguridad en: un equipo, en una red de equipos LAN (*Local Área Network*) o WLAN (*Wireless local Área Network*), aplicaciones Web entre otros, por medio de ataques informáticos simulados idénticos a los que realizaría un *Cracker* o *Black Hat Hacker* pero sin poner en riesgo la información o la disponibilidad de los servicios, esto se hace con el fin de encontrar las posibles amenazas en los sistemas IT antes de que las descubra un atacante (externo o interno).

Los ataques más relevantes a los que pueden estar expuestos los sistemas web son los siguientes:

El ***Sql Injection*** es un ataque en el que se inserta código malicioso en cadenas que luego se pasan a una instancia de la Base de datos (puede ser SQL Server, Mongo DB u otra) para su análisis y ejecución. Cualquier procedimiento que construya declaraciones SQL debe revisarse para detectar vulnerabilidades de inyección porque la base de datos ejecutará todas las consultas que reciba y sean sintácticamente válidas. Incluso los datos parametrizados pueden ser manipulados por un atacante hábil y decidido [2].

La **Inyección de comandos** es un ataque en el que el objetivo es la ejecución de comandos arbitrarios en el sistema operativo host a través de una aplicación vulnerable. Los ataques de inyección de comandos son posibles cuando una aplicación pasa datos no seguros proporcionados por el

usuario (formularios, cookies, encabezados HTTP, etc.) a un shell del sistema. En este ataque, los comandos del sistema operativo proporcionados por el atacante generalmente se ejecutan con los privilegios de la aplicación vulnerable. Los ataques de inyección de comandos son posibles en gran parte debido a una validación de entrada insuficiente [3].

Los ataques de ***Cross Site Scripting (XSS)*** son un tipo de inyección, en el que se inyectan scripts maliciosos en sitios web. Los ataques XSS ocurren cuando un atacante usa una aplicación web para enviar código malicioso, generalmente en forma de un script del lado del navegador, a un usuario final diferente. Un atacante puede usar XSS para enviar un script malicioso a un usuario desprevenido. El navegador del usuario final no tiene forma de saber que no se debe confiar en el script y lo ejecutará. Debido a que cree que el script proviene de una fuente confiable, el script malicioso puede acceder a las cookies, tokens de sesión u otra información confidencial retenida por el navegador y utilizada con ese sitio. Estos scripts pueden incluso reescribir el contenido de la página HTML [4].

El ataque de **Denegación de Servicio *(DoS)*** se centra en hacer que un recurso (sitio, aplicación, servidor) no esté disponible para el propósito para el que fue diseñado. Hay muchas formas de hacer que un servicio no esté disponible para usuarios legítimos mediante la manipulación de paquetes de red, programación, lógica o recursos que manejan vulnerabilidades, entre otras. Si un servicio recibe una gran cantidad de solicitudes, puede dejar de estar disponible para usuarios legítimos. De la misma manera, un servicio puede detenerse si se explota una vulnerabilidad de programación o la forma en que el servicio maneja los recursos que utiliza [5].

Figura 3 - WAF en la nube.
https://cso.computerworld.es/tendencias/las-soluciones-waf-en-la-nube-cogen-fuerza-en-el-mercado

Los ***Web Application Firewall (WAF)*** en la nube se han convertido en uno de los productos estrella del mercado de la seguridad a la hora de proteger las aplicaciones web públicas. Para 2020, asegura Gartner [6], los de **hardware independiente** representarán menos del 20% de las nuevas implementaciones de WAF, en comparación con el 40% actual, lo que **significa que más del 50% de las aplicaciones web públicas estarán bajo el paraguas de servicios WAF en la nube**.

Entre sus casos de uso, destacan por su protección frente a ataques y explotaciones de vulnerabilidades en códigos personalizados o de terceros. Estos servicios también suelen incluir defensa ante **DDoS y mitigación básica de *bots* así como control de reputación o huella digital** de dispositivos.

Las soluciones WAF trabajan en la capa de aplicación (capa 7 modelo OSI), interceptando y analizando las peticiones realizadas a los sistemas web contra un sistema de reglas, con el objeto de impedir que una petición maliciosa llegue al mismo.

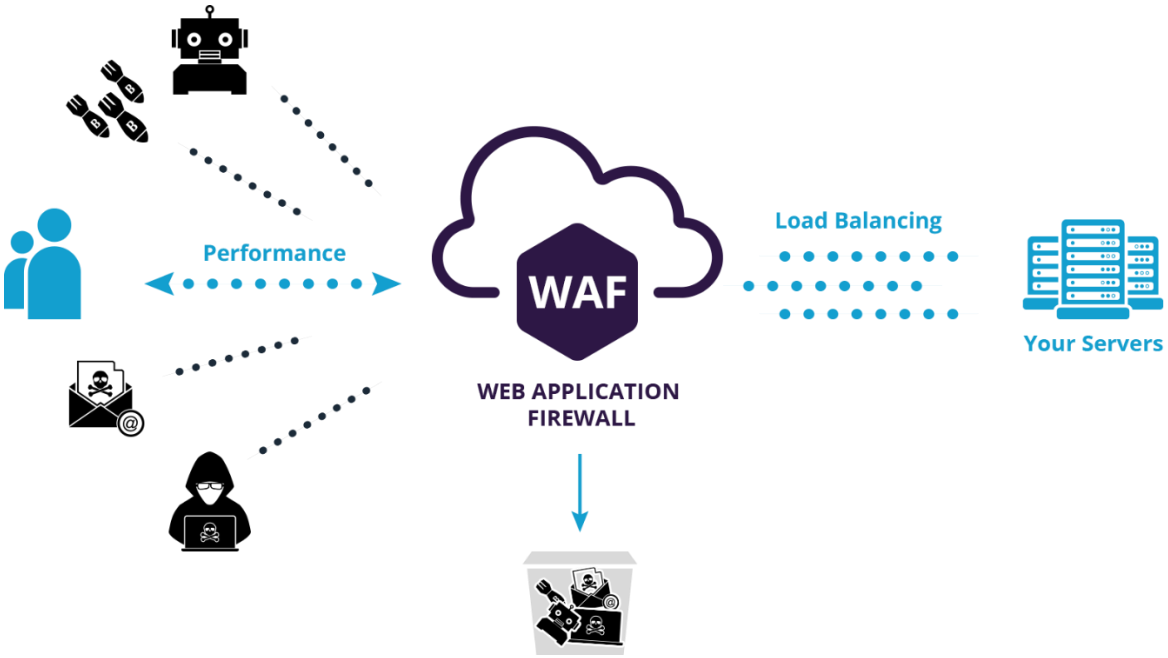


Figura 4 - WAF. *Aplicación para protección contra ataques*
https://host.com.tw/advanced-AI-WAF

Básicamente, todas las soluciones WAF funcionan de manera similar: actúan como un servicio intermediario entre la aplicación de un sitio web y el visitante que navega por ese sitio web, interceptando y eliminando solicitudes maliciosas antes de que puedan causar daños. La diferencia real viene en cómo se despliegan dentro de la arquitectura de la red. Las tecnologías WAF no pretenden reemplazar los controles existentes, sino complementarlos.

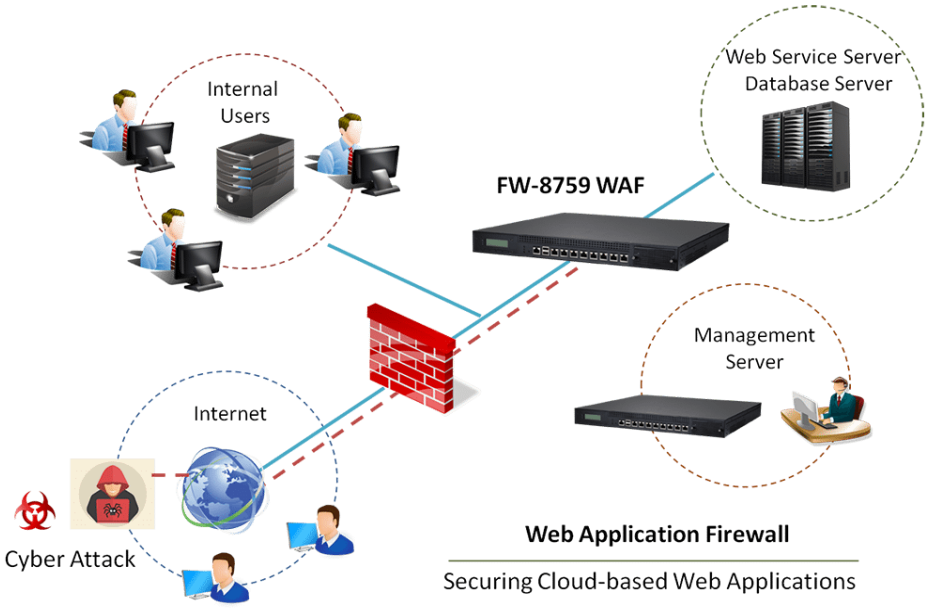


Figura 5 - Arquitectura Firewall + WAF.

https://www.lanner-america.com

2) Principales razones para usar WAF

Una de las principales razones para utilizar un WAF es la **protección continua**, evita ataques automáticos dirigidos por *spambots* o *bots* que se programan para buscar vulnerabilidades en las webs de internet. Sus ataques suelen ser de infecciones o phishing.

También se puede **ahorrar recursos y dinero** con la ayuda de un WAF, podrás parar casi todos los intentos de ataque hacia las aplicaciones web y, a su vez, entenderás las peticiones maliciosas que los ciberdelincuentes utilizan para atacar, y de esa manera, poder brindarles una mejor protección a tus aplicaciones web. Evitando que la aplicación web quede fuera de servicio por un ataque *DDoS*.

Otra cosa que se debe tener en cuenta es la **protección ante lo inevitable**, pues minimiza los riesgos de filtración de datos y fugas de información cuando no sea posible realizar cambios inmediatos en el código de la aplicación.

Por último, pero no menos importante, la **visibilidad**, identifica todos los contenidos de las aplicaciones y aplica políticas de protección automáticas y personalizadas. Gracias a entender tus servicios y activos, aplica medidas para mejorar el rendimiento y la seguridad de tus aplicaciones.

3) Principales riesgos al usar *WAF*

Uno de los principales riesgos al utilizar un *WAF* es que se debe de tener en cuenta que su uso posiblemente requiera cambios en la infraestructura de red.

El “OWASP TOP 10” brinda un sistema de reglas que cubren los principales ataques, pero cada empresa deberá crear o modificar algunas de esas reglas para mejorar la protección contra los ataques. Si las reglas no están bien definidas puede haber falsos positivos que pueden llegar a tener un impacto en el negocio.

Se deberá capacitar sobre el uso y manejo del WAF al personal encargado del mismo.

Cada versión del sistema web debe de contar con un WAF.

4) Despliegue de los WAF

Existen tres opciones de despliegue para los *WAF*, una de las más comúnmente utilizadas es la llamada “***In-line Appliance Firewalls”***, el *WAF* se despliegan en la red de su organización, tradicionalmente dentro de la *DMZ* (conocida también como zona desmilitarizada, como sus siglas en ingles *DeMilitarized Zone*).

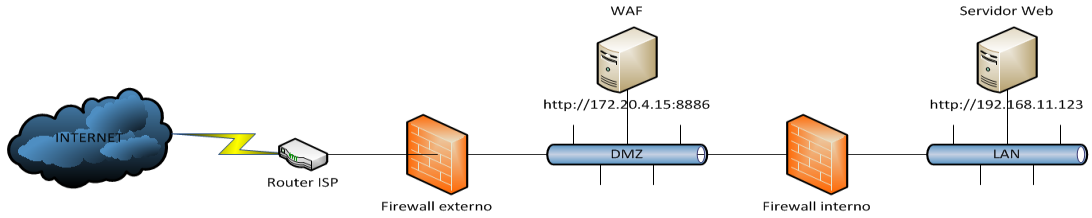


Fig.6 WAF en una DMZ.

Otra opción es la llamada “***End-point Firewalls***”, aquí se despliegan dentro del servidor de alojamiento. Se pueden implementar en diferentes niveles: en el sistema operativo (por ejemplo, IDS/IPS), en el servidor web (es decir, Apache) y en la aplicación (por ejemplo, WordPress, Drupal).

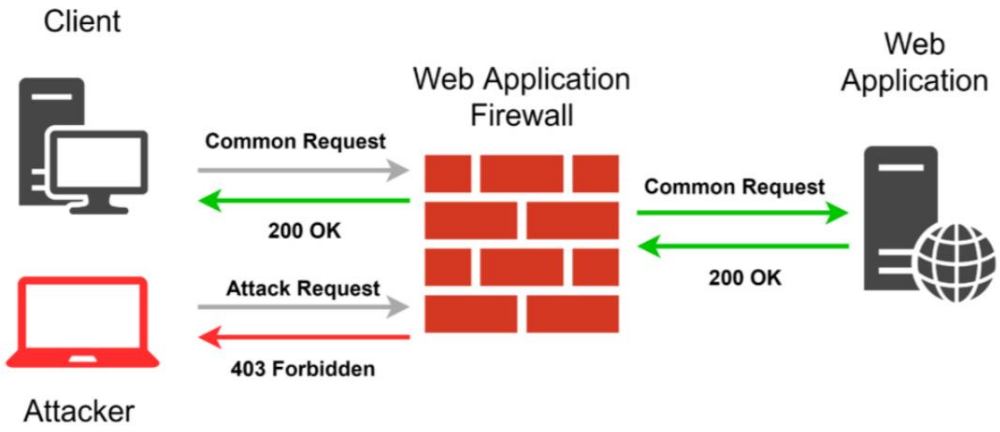


Figura 7 - WAF cómo endpoint.

https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/26225/1/TIB20.pdf

La última opción es la denominada "***Cloud-based Firewalls***", aquí los WAF se despliegan en la nube, fuera de su entorno de alojamiento.

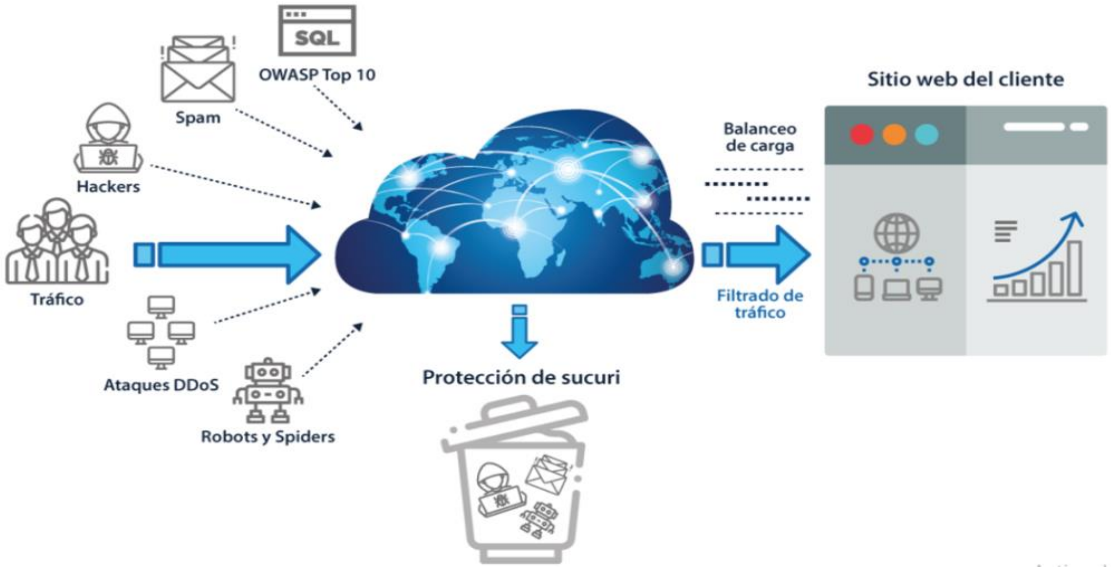


Figura 8 - WAF basado en la nube.
https://deepsecurity.pe/waf.html

5) Soluciónes *WAF Open Source*

5.1) *ModSecurity*

ModSecurity es un juego de herramientas para el monitoreo, aplicación y control de acceso de aplicaciones web en tiempo real. Me gusta pensar que es un facilitador: no hay reglas estrictas que le digan qué hacer; en cambio, depende de usted elegir su propio camino a través de las funciones disponibles. Es por eso que nos preguntamos qué puede hacer *ModSecurity*, y no qué hace.

5.1.1) Escenarios de uso

A continuación, se detalla una lista de los escenarios de uso más importantes:

1. ***Monitoreo de seguridad de aplicaciones en tiempo real y control de acceso:*** En esencia, *ModSecurity* le brinda acceso al flujo de

tráfico HTTP en tiempo real, junto con la capacidad de inspeccionarlo. Esto es suficiente para el monitoreo de seguridad en tiempo real. Hay una dimensión adicional que posibilita a través del mecanismo de almacenamiento persistente de *ModSecurity* rastrear elementos del sistema a lo largo del tiempo y realizar la correlación de dichos eventos. Puede bloquear accesos de manera confiable, si así lo desea, pues *ModSecurity* utiliza el almacenamiento completo de solicitudes y respuestas en el búfer.

2. ***Registro completo de tráfico HTTP:*** *ModSecurity* le brinda la capacidad de registrar cualquier cosa que necesite, incluidos los datos de transacciones sin procesar, que es esencial para el análisis forense. Además, puede elegir qué transacciones se registran, qué partes de una transacción se registran y qué partes se desinfectan.

3. ***Evaluación continua de seguridad pasiva:*** La evaluación de seguridad se ve en gran medida como un evento programado activo, en el que se busca un equipo independiente para intentar realizar un ataque simulado. La evaluación de seguridad pasiva continua es una variación del monitoreo en tiempo real, donde, en lugar de enfocarse en el comportamiento de las partes externas, usted se enfoca en el comportamiento del sistema mismo. Es un sistema de alerta temprana que puede detectar rastros de muchas anomalías y debilidades de seguridad antes de que sean explotadas.

4. ***Fortalecimiento de la aplicación web:*** Uno de los usos de *ModSecurity* es la reducción de la superficie de ataque, en la que se restringe selectivamente las características HTTP que está dispuesto a aceptar (por ejemplo, métodos de solicitud, encabezados de solicitud, tipos de contenido, etc.). *ModSecurity* puede ayudarlo a aplicar muchas restricciones similares, ya sea directamente o mediante la colaboración con otros módulos de Apache. Por ejemplo, es posible solucionar muchos problemas de administración de

sesiones, así como vulnerabilidades de falsificación de solicitudes entre sitios.

5. ***Algo pequeño, pero muy importante:*** La vida real a menudo nos genera demandas inusuales, y es cuando la flexibilidad de *ModSecurity* es útil donde más la necesita. Puede ser una necesidad de seguridad, pero también puede ser algo completamente diferente. Por ejemplo, algunas personas usan *ModSecurity* como un enrutador de servicios web XML, combinando su capacidad de analizar *XML* y aplicar expresiones *XPath* con su capacidad de proxy.

5.1.2) Principios en los que se basa *ModSecurity*

Existen cuatro principios en los que se basa *ModSecurity*. El primero de ellos es la **Flexibilidad**, *ModSecurity* logra flexibilidad al brindarle un poderoso lenguaje de reglas, que le permite hacer exactamente lo que necesita, en combinación con la capacidad de aplicar reglas solo donde lo necesita.

También está la **Pasividad**, *ModSecurity* tendrá mucho cuidado de nunca interactuar con una transacción a menos que se lo indique. “Eso es simplemente porque no confío en las herramientas, incluso en la que construí, para tomar decisiones por mí”. Es por eso que *ModSecurity* le dará mucha información, pero finalmente le dejará las decisiones a usted.

Luego tenemos la **Previsibilidad**, no existe una herramienta perfecta, pero la mejor opción es una previsible. Conociendo bien a *ModSecurity*, podemos comprender sus puntos débiles y solucionarlos.

Por último, tenemos la **Calidad sobre cantidad**, se nos ocurrieron muchas ideas sobre lo que *ModSecurity* podría hacer. No actuamos sobre la mayoría de ellas. Las guardamos para más tarde. ¿Por qué? Porque entendimos que tenemos recursos limitados disponibles a nuestra disposición y que nuestras mentes (ideas) son mucho más rápidas que nuestras habilidades de implementación. Elegimos limitar la funcionalidad disponible, pero hicimos realmente bien lo que decidimos mantener.

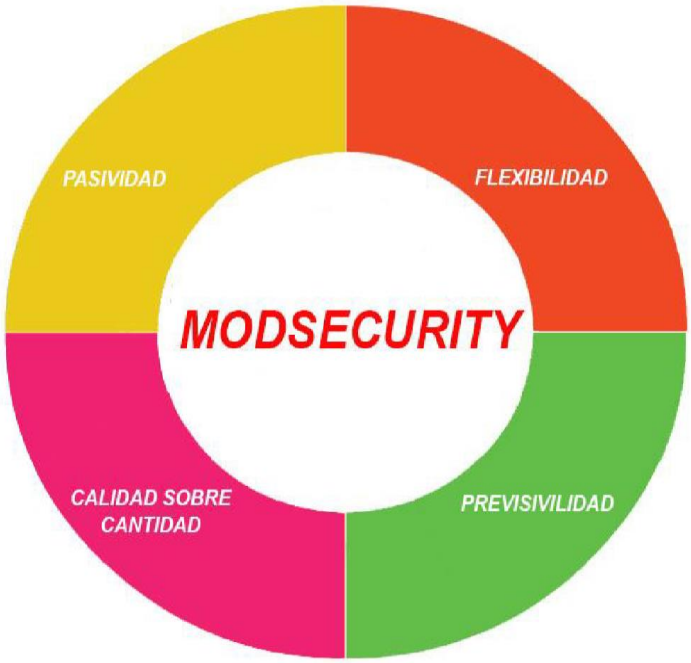


Fig.9 – Principios de *ModSecurity*

Hay bits en *ModSecurity* que quedan fuera del alcance de estos cuatro principios. Por ejemplo, *ModSecurity* puede cambiar la forma en que Apache se identifica con el mundo exterior, encapsular el proceso de Apache e incluso implementar un esquema elaborado para lidiar con una vulnerabilidad *XSS* universal que alguna vez fue infame en Adobe Reader. Por todo esto podemos decir que *ModSecurity* es una herramienta confiable y predecible que permite la inspección de tráfico HTTP.

5.1.3) Opciones de implementación

ModSecurity admite dos opciones de implementación: implementación de proxy inverso e integrado (*embedded*). No hay una forma correcta de usarlos, por lo tanto, deberá elegir una opción basada en lo que mejor se adapte a sus circunstancias. Hay ventajas y desventajas en ambas opciones:

Cuando lo implementamos **Integrado** *(embedded)*, debido a que *ModSecurity* es un módulo de Apache, puede agregarlo a cualquier versión

compatible del mismo. Por el momento eso significa una versión de Apache razonablemente reciente de la rama 2.0.x, aunque se recomienda una versión más reciente 2.2.x o superior. La implementación “integrada” es una gran opción para aquellos que ya tienen su arquitectura diseñada y no quieren cambiarla. Esta implementación también es la única opción si necesita proteger cientos de servidores web. En tales situaciones, no es práctico construir una capa de seguridad separada basada en proxy. *Embedded ModSecurity* no solo no introduce nuevos puntos de falla, sino que se escala sin problemas a medida que aumenta la infraestructura web subyacente. El principal desafío con la implementación integrada es que los recursos del servidor se comparten entre el servidor web y *ModSecurity*.

Como **Proxy inverso** *(reverse proxy)*, los proxys inversos son efectivos enrutadores HTTP, diseñados para interponerse entre los servidores web y sus clientes. Cuando instala un proxy inverso dedicado en Apache y le agrega *ModSecurity*, obtiene un firewall de aplicación web de red "adecuado", que puede ser usado para proteger cualquier número de servidores web en la misma red. Muchos profesionales de la seguridad prefieren tener una capa de seguridad separada. Por esto, con él se obtiene un aislamiento completo de los sistemas que se están protegiendo. En referencia al rendimiento, un *ModSecurity* independiente tendrá recursos dedicados, lo que significa que podrá hacer más (es decir, tener reglas más complejas). La principal desventaja de este enfoque es el nuevo punto de falla, que deberá abordarse con una configuración de alta disponibilidad de dos o más servidores proxy inversos.

ModSecurity es una herramienta muy buena, pero hay una serie de características, grandes y pequeñas, que podrían agregarse. Las pequeñas características son las que le facilitarían la vida con *ModSecurity*, tal vez automatizando parte del trabajo aburrido (por ejemplo, un bloqueo persistente, que ahora tiene que hacer manualmente). Pero en realidad solo hay dos características que llamaría faltantes:

Aprendizaje

Defender las aplicaciones web es difícil, porque hay muchas y todas son diferentes. Cada aplicación web crea efectivamente su propio protocolo de comunicación, por lo tanto, sería muy útil que *ModSecurity* observara el tráfico de la aplicación y creara un modelo que luego podría usarse para generar políticas o ayudar con falsos positivos.

Modo pasivo de despliegue

ModSecurity solo puede integrarse en Apache 2.x, pero cuando lo implementa como proxy inverso, puede usarse para proteger cualquier servidor web. Sin embargo, los servidores proxy inversos no son del agrado de todos, y a veces sería muy útil implementar *ModSecurity* pasivamente, sin tener que cambiar nada en la red.

(ModSecurity.org, 2020. https://modsecurity.org/about.html).

ModSecurity es un conjunto de herramientas flexible, sin reglas estrictas y rápidas que le indicaran cómo debe ser utilizado. Por lo general, *ModSecurity* le deja la libertad de decidir cómo aprovechar las funciones que tiene disponibles. Esta flexibilidad es un elemento central de la identidad de *ModSecurity* y complementa su estructura de código abierto. De hecho, puede disfrutar de un acceso completo a su código fuente, lo que le permite a cualquier organización personalizar la herramienta para que se adapte a sus necesidades específicas. *ModSecurity* es una creación versátil ideal para numerosos escenarios de uso, y eso es crucial para cualquiera que quiera herramientas que les permitan lograr lo que tienen que hacer con restricciones mínimas.

5.2) Shadow Daemon

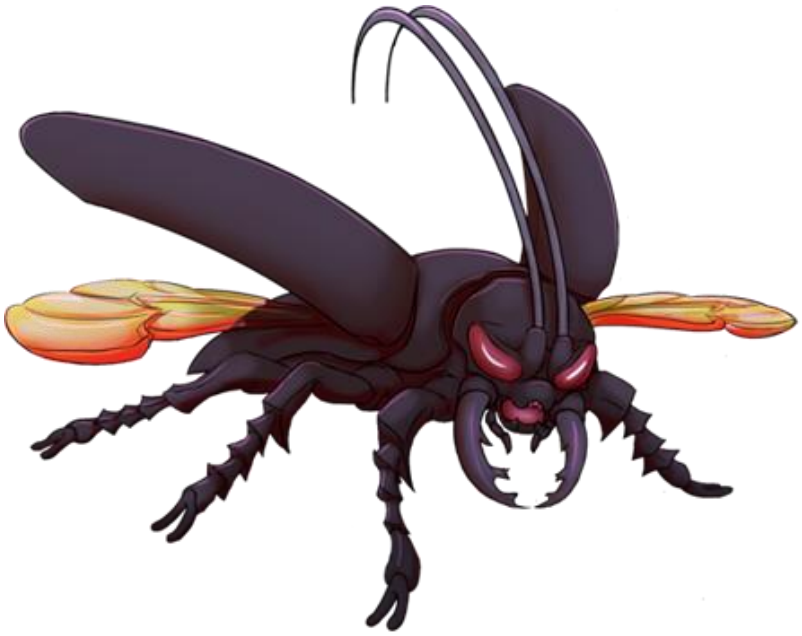


Figura 10 - Shadow Daemon
https://github.com/zecure/shadowd

Shadow Daemon es una colección de herramientas para detectar, registrar y bloquear ataques a aplicaciones web. Técnicamente hablando, Shadow Daemon es un firewall de aplicaciones web que intercepta solicitudes y filtra los parámetros maliciosos. Es un sistema modular que separa la aplicación web, el análisis y la interfaz para aumentar la seguridad, la flexibilidad y la capacidad de expansión. A diferencia de otros firewalls, *Shadow Daemon* capta las solicitudes a nivel de aplicación, y no a nivel de protocolo.

Shadow Daemon es un software gratuito. Se publica bajo la licencia GPLv2 [7], por lo que es de código abierto y el código puede ser examinado, modificado y distribuido por todos. Es de fácil uso y fácil de instalar, y se puede administrar a trasvés de una interfaz web clara y estructurada que le permite examinar los ataques con gran detalle. Dicha interfaz también viene con *scripts* de *shell* que se pueden usar para enviar informes semanales por correo electrónico, rotar los registros, etc.

5.2.1) Características de Shadow Daemon

Detección Precisa

Posee una detección precisa de solicitudes maliciosas al combinar listas negras, listas blancas y verificación de integridad. La lista negra utiliza sofisticadas expresiones regulares para buscar patrones de ataque conocidos en la entrada del usuario. La lista blanca, por otro lado, busca irregularidades en la entrada del usuario basándose en reglas estrictas que definen cómo debería verse dicha entrada. La verificación de integridad compara las sumas de verificación criptográficamente seguras de los scripts ejecutados con valores predefinidos. Juntos pueden detectar casi cualquier ataque a una aplicación web y aun así tener una tasa de falsos positivos muy baja.

Shadow Daemon es capaz de detectar ataques comunes como inyecciones SQL, inyecciones XML, inyecciones de código, inyecciones de comando, *cross-site scripting*, *backdoor Access* y más.

A diferencia de muchos otros firewalls de aplicaciones web, si es posible, Shadow Daemon no bloquea completamente las solicitudes maliciosas, en su lugar, solo filtra las partes peligrosas de una solicitud y la deja continuar después. Esto hace que los ataques sean imposibles, pero no frustra innecesariamente a los visitantes en el caso de falsos positivos.

Arquitectura segura

Posee una arquitectura segura pues Shadow Daemon está más cerca de la aplicación que la mayoría de los demás firewalls de aplicaciones web. Recibe exactamente la misma entrada que recibe la aplicación web y, por lo tanto, es casi imposible evitar la detección ocultando el ataque. Sin embargo, las partes más complejas de Shadow Daemon están separadas de la aplicación web para garantizar un cierto estándar de seguridad.

Interfaz de usuario

Shadow Daemon posee una interfaz de usuario amigable, para utilizarla solo debe ir a la interfaz web e iniciar sesión con su usuario administrador. De forma predeterminada, puede acceder a él en el puerto 8080, por ejemplo, http://127.0.0.1:8080 si se ejecuta en localhost.

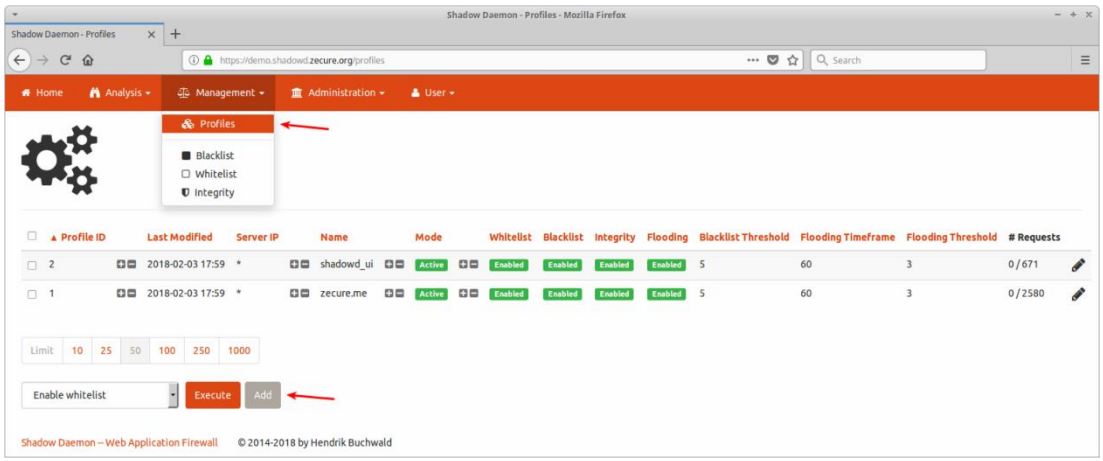


Figura 11 – Interfaz de usuario.

https://shadowd.zecure.org/overview/user_interface/

Aquí podrán agregar, modificar o eliminar perfiles, configurar las IP de los conectores, configurar el modo de trabajo (activo o pasivo), habilitar o deshabilitar las listas blanca y negra y mucho más. La interfaz de usuario es imprescindible para la configuración de *Shadow Daemon*. También se puede utilizar para analizar ataques, aprender sobre los atacantes y mejorar las reglas en función de ese nuevo conocimiento.

5.2.2) Arquitectura de *Shadow Daemon*

El siguiente diagrama representa el diseño y la comunicación de *Shadow Daemon*.

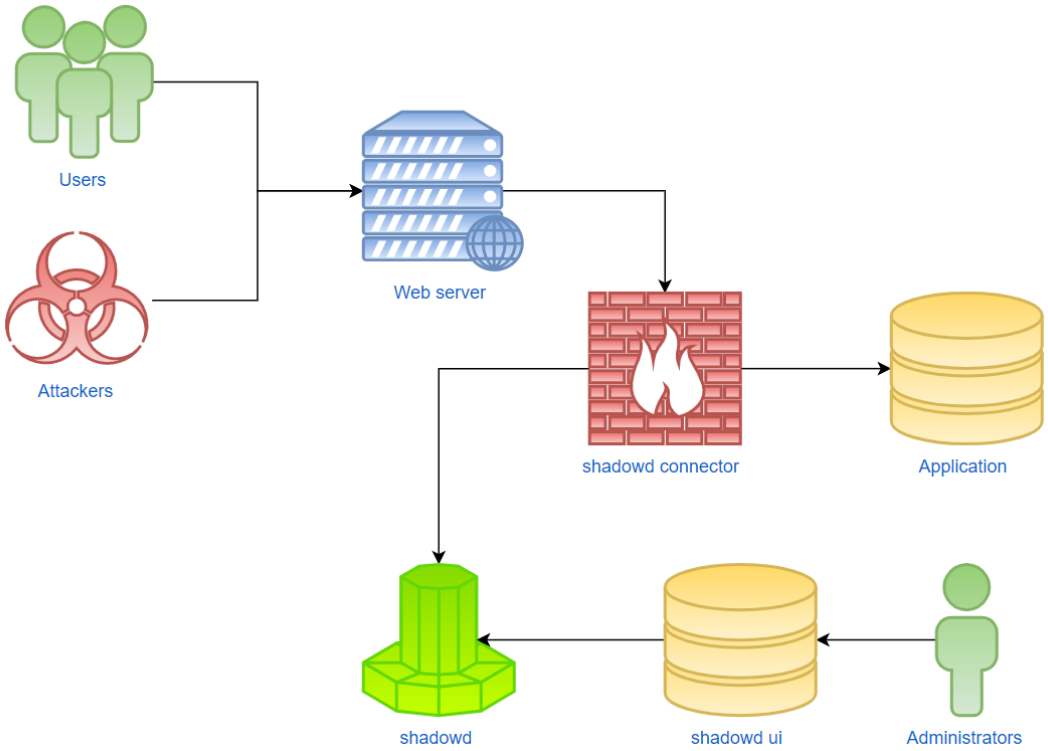


Fig. 12 – Arquitectura de Shadow Daemon.
https://shadowd.zecure.org/documentation/architecture/

El conector se ejecuta cada vez que un cliente solicita un recurso. Establece una conexión TCP con el servidor *shadowd* y transmite la IP del cliente, el recurso, la entrada del usuario y las sumas de verificación. El servidor procesa y analiza los datos con la lista negra, la lista blanca y la verificación de integridad y devuelve los identificadores de entrada peligrosa. El conector usa los identificadores para desactivar todas las amenazas y cargar el recurso solicitado originalmente.

5.2.3) Identificación de solicitudes maliciosas

Shadow Daemon utiliza 3 métodos para identificar solicitudes maliciosas, estos son la lista negra (*blacklist*), la lista blanca (*whitelist*) y el algoritmo de integridad (*integrity*). Estos métodos utilizan reglas que especifican como deberían verse las entradas o peticiones al sitio web. Por lo tanto, antes de

describir los métodos utilizados para la identificación de solicitudes maliciosas es conveniente realizar una descripción de las reglas.

Reglas

Shadow Daemon requiere buenas reglas para funcionar correctamente. Las reglas difieren de una aplicación a otra, por lo que no existe una solución universal. Crear reglas no es difícil, pero antes de comenzar a invertir tiempo en la creación de nuevas reglas debemos verificar si alguien más ya creó reglas para su aplicación. Existe un repositorio oficial de las reglas de Shadow Daemon que se encuentra en *zecure/shadowd_rules* [8], aquí los colaboradores de *Shadow Daemon* guardan las reglas creadas para sus aplicaciones y que pueden ser de utilidad para los demás, pues cuanta más gente haga esto, menos trabajo será configurar correctamente el cortafuegos.

Las reglas se pueden importar a través de la interfaz web, pero deben ser testeadas antes de activar la protección, porque podría haber desajustes causados por complementos, diferentes idiomas y cosas similares.

Reglas de la lista negra (*blacklist rules*)

Las reglas de la lista negra se utilizan para sobrescribir el umbral global para una entrada de usuario específica.

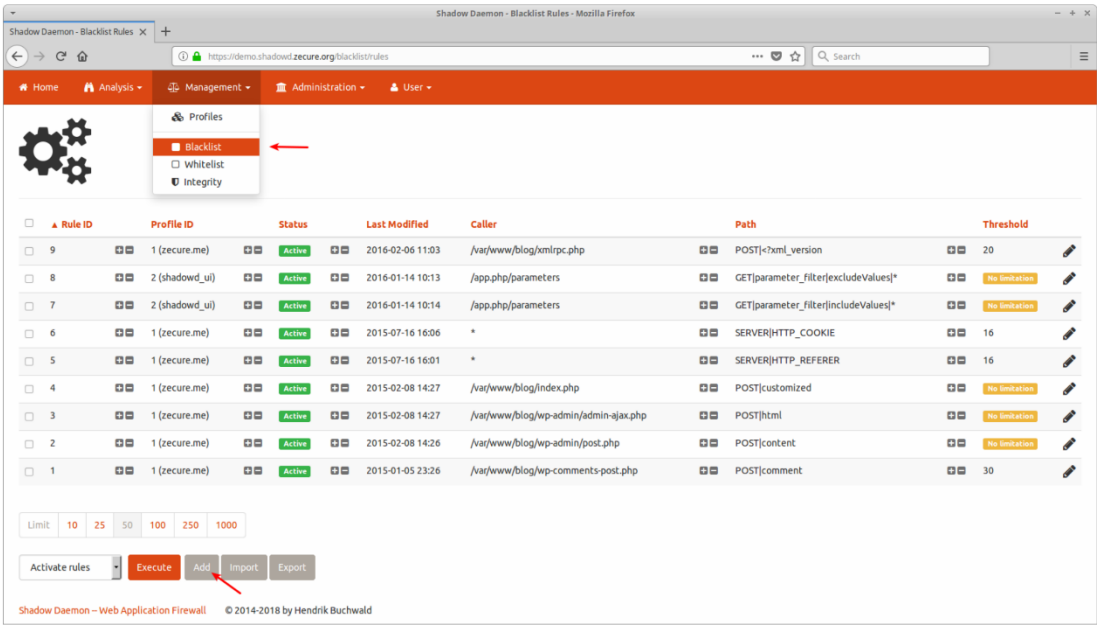


Figura 13 – Lista negra.

https://shadowd.zecure.org/tutorials/rules/

Esto es necesario pues cuanto más contenido contenga la entrada del usuario, más probable será que la lista negra detecte ataques inexistentes. Por ejemplo, la lista negra detecta ciertas combinaciones de palabras que se utilizan para inyecciones de SQL, pero las mismas combinaciones podrían ocurrir en un texto normal. La lista negra busca “UNION ... SELECT y SELECT ... FROM” para detectar ataques como “UNION SELECT name, pass FROM users”, pero los mismos patrones coincidirían con el texto " *Start an union and select representatives from the employees* ". Para compensar esto, puede agregar excepciones a los parámetros en una aplicación web que contienen grandes cantidades de entradas de usuario y, por lo tanto, es más probable que generen falsos positivos. De esta manera, puede tener un umbral global bajo para asegurarse de que incluso los ataques ofuscados se detecten la mayor parte del tiempo y aún eliminen gran cantidad de falsos positivos.

Reglas de la lista blanca (*whitelist rules*)

Las reglas de la lista blanca se utilizan para definir qué entrada de usuario está permitida y cómo debe verse esta entrada.

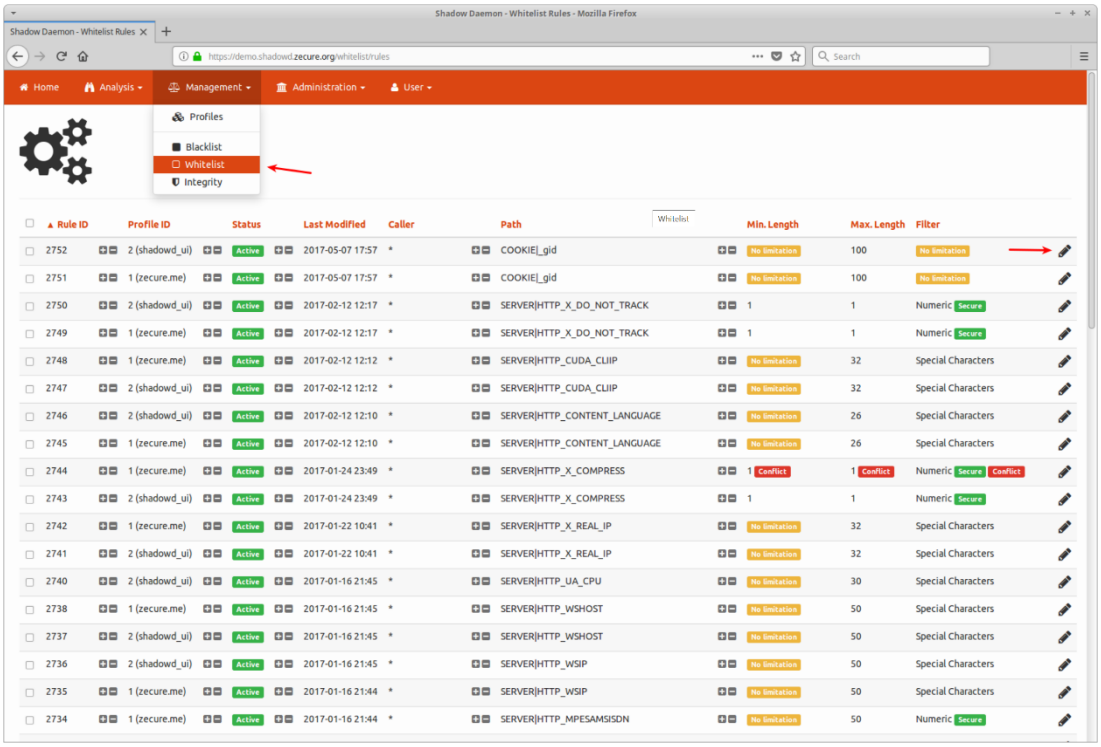


Figura 14 – Lista blanca.

https://shadowd.zecure.org/tutorials/rules/

Este enfoque tiene una gran ventaja en comparación con la lista negra, pero también una gran desventaja. Por un lado, es imposible pasar por alto el control si las reglas son estrictas. No necesita información sobre técnicas de ataque y nunca pasa de moda. Por otro lado, puede ser mucho trabajo configurarlo. Las aplicaciones web modernas tienen muchos parámetros y cada uno de ellos requiere una regla correcta. Debido a esto, no se recomienda agregar las reglas a mano. En su lugar, es conveniente usar el *generador* [9] para crear reglas a partir de los datos de aprendizaje y solo corregir las reglas incorrectas o faltantes a mano.

Reglas de integridad (*integrity rules*)

Las reglas de integridad se utilizan para definir qué sumas de comprobación están permitidas y cuáles deberían ser estas sumas de comprobación, por lo que también se utiliza un enfoque de listas blancas.

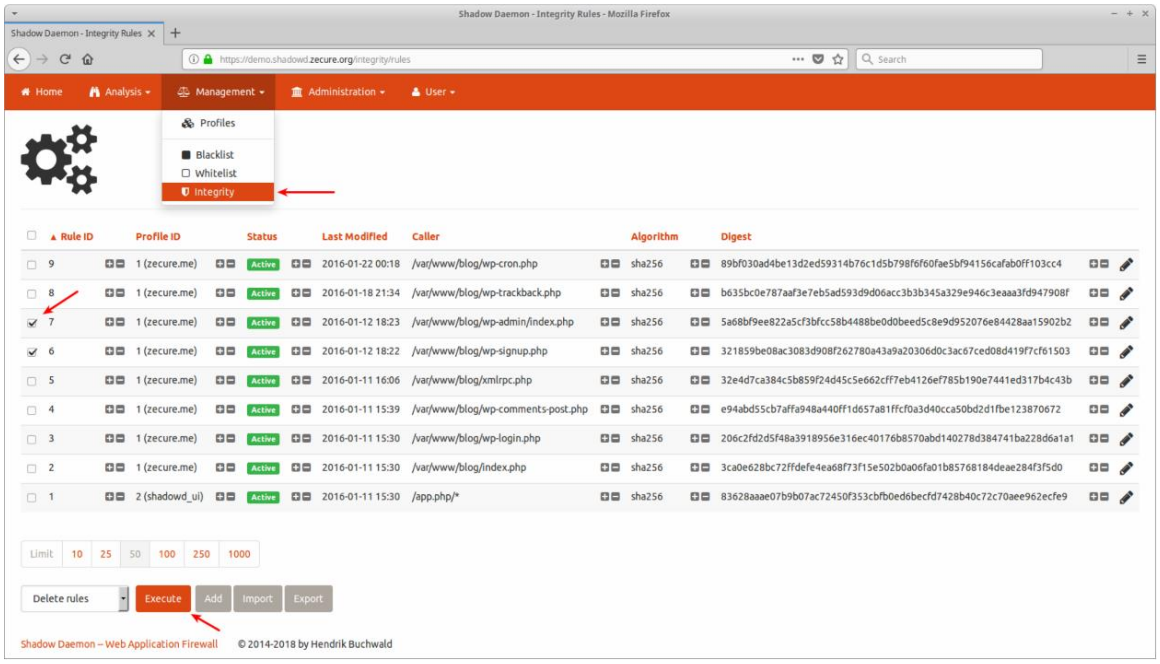


Figura 15 – Integridad.

https://shadowd.zecure.org/tutorials/rules/

Esta es la razón por la que se recomienda utilizar el *generador* para crear reglas a partir de los datos de aprendizaje.

Lista Negra (*Blacklist*)

El algoritmo de lista negra busca patrones de ataque conocidos en la entrada del usuario.

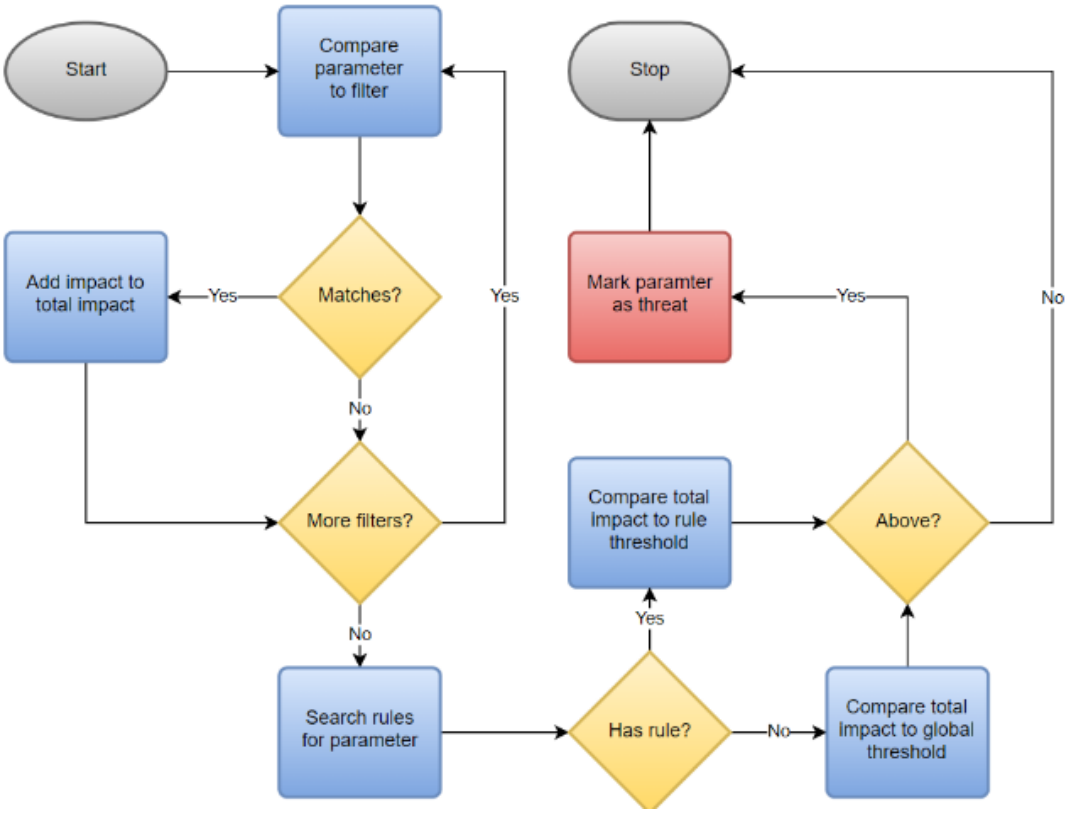


Figura 16 – Diseño de la lista negra.

https://shadowd.zecure.org/documentation/blacklist/

Utiliza expresiones regulares para identificar patrones de ataque conocidos. Cada filtro tiene un impacto numérico que intenta especificar la peligrosidad y su falta de ambigüedad. Los impactos de todos los filtros coincidentes se agregan y comparan con un umbral. Si el impacto total es mayor que el umbral, la entrada se clasifica como una amenaza. Hay que tener en cuenta que una lista negra no garantiza una seguridad perfecta. Es buena para detectar la mayoría de los patrones de ataque comunes, pero siempre habrá técnicas que no conoce y, por lo tanto, no detecta. Una vez que los atacantes pueden ejecutar su propio código, tienen muchas formas de ofuscar sus cargas útiles y, por lo tanto, disminuir aún más el valor de impacto. Para compensar esto, hay que asegurarse de utilizar un umbral de impacto global muy bajo y, si es necesario, incrementarlo.

Lista Blanca (*Whitelist*)

El algoritmo de lista blanca compara la entrada del usuario con las reglas que especifican cómo debería verse la entrada.

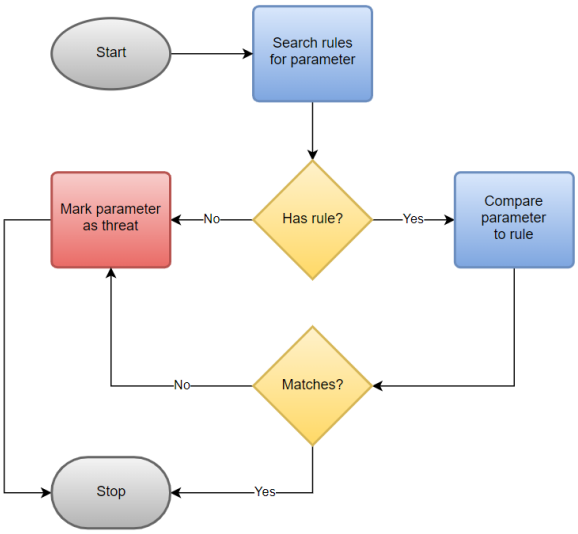


Figura 16 – Diseño de la lista blanca.

https://shadowd.zecure.org/documentation/whitelist/

Primero verifica si el parámetro tiene una regla. El término lista blanca implica que cada parámetro sin al menos una regla de coincidencia se considera una amenaza. Si hay reglas, el algoritmo verifica si tienen restricciones de longitud y si se cumplen. Finalmente, el algoritmo prueba el conjunto de caracteres de la entrada con la ayuda de expresiones regulares.

Los siguientes filtros de juego de caracteres para la lista blanca están disponibles (no distinguen entre mayúsculas y minúsculas):

Name	Characters
Numeric	0123456789
Numeric (Extended)	-0123456789.,
Hexadecimal	0123456789abcdef
Alphanumeric	0123456789abcdefghijklmnopqrstuvwxyz
Base64	0123456789abcdefghijklmnopqrstuvwxyz=+/\s
Special Characters	0123456789abcdefghijklmnopqrstuvwxyz.,:-+_\s
Everything	*Absolutely everything*

Figura 17 – Filtro de caracteres.

https://shadowd.zecure.org/documentation/whitelist/

Todas las entradas de usuario deben comprobarse con este filtro.

Integridad (*Integrity*)

El algoritmo de integridad compara las sumas de comprobación criptográficamente seguras del script ejecutado con reglas que especifican cuáles deberían ser estas sumas de comprobación.

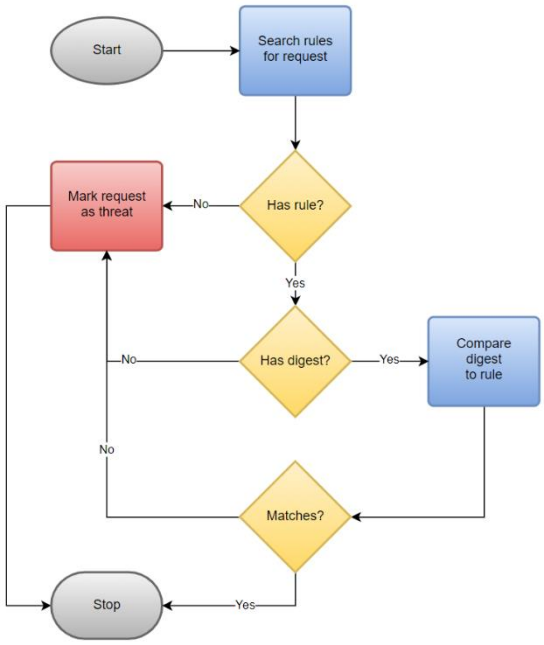


Figura 18 – Diseño del algoritmo de integridad.
https://shadowd.zecure.org/documentation/integrity/

El algoritmo de integridad también utiliza un enfoque de listas blancas. El término lista blanca implica que cada solicitud requiere una regla de coincidencia, de lo contrario, se considera una amenaza. Primero verifica si la solicitud tiene una regla. Si hay una regla, el algoritmo verifica si la solicitud tiene un resumen. Finalmente, el algoritmo prueba si el resumen de la solicitud coincide con el resumen de la regla. Hay que tener en cuenta que la verificación de integridad solo valida la integridad del script ejecutado, pero no la integridad del código incluido dinámicamente. El propósito de esta comprobación es evitar la ejecución de scripts no deseados, no hacer que las inyecciones de código sean completamente imposibles.

(Shadowd.zecure.org,2021. https://shadowd.zecure.org/overview/introduction/)

5.2.4) Corolario

Por todo lo antes comentado, vemos que *Shadow Daemon* es un *WAF* con muchas funcionalidades, que posee la documentación online necesaria para instalarlo y configurarlo, y que además tiene un set de reglas probadas que podría brindarle la protección necesaria a cualquier sitio web. Lo que nos queda por preguntar es ¿quién debería usar *Shadow Daemon*?, pues *Shadow Daemon* es para personas, empresas u organisaciones que desean ejecutar su propio sitio web dinámico sin tener que preocuparse constantemente por ataques y vulnerabilidades, o saber si su sitio web es atacado y cómo lo hacen, y también es para aquellos que no quieren depositar ciegamente su confianza en software de código cerrado que hace su trabajo en secreto y cuesta una fortuna.

5.3) NAXSI

Figura 19 – NAXSI.
https://server-talk.tistory.com/305

NAXSI es un *Web Application Firewall* de código abierto, de alto rendimiento y de bajo mantenimiento para NGNIX [10]. El acrónimo significa Nginx Anti XSS y SQL Injection.

Técnicamente, es un módulo de Nginx disponible como paquete para muchas plataformas tipo UNIX. De manera predeterminada, lee un pequeño

subconjunto de reglas sencillas (y legibles) que contienen el 99% de los patrones conocidos involucrados en las vulnerabilidades web. Por ejemplo, <, | o *drop* los cuales se supone que no son parte de una URI [11].

Siendo muy simples, esos patrones podrían coincidir con consultas legítimas, es deber del administrador de Naxsi agregar reglas específicas que incluyan en la lista blanca los comportamientos legítimos. El administrador puede agregar listas blancas de dos maneras, la primera es realizándolo manualmente analizando el registro de errores de Nginx, o como segunda opción y la más recomendada es iniciar el proyecto con una fase de aprendizaje automático intensivo que generará automáticamente reglas de listas blancas con respecto al comportamiento del sitio web.

En resumen, Naxsi se comporta como un firewall que *"dropeará"* todo por defecto, la única tarea es agregar las reglas de ACEPTAR requeridas para que el sitio web de destino funcione correctamente.

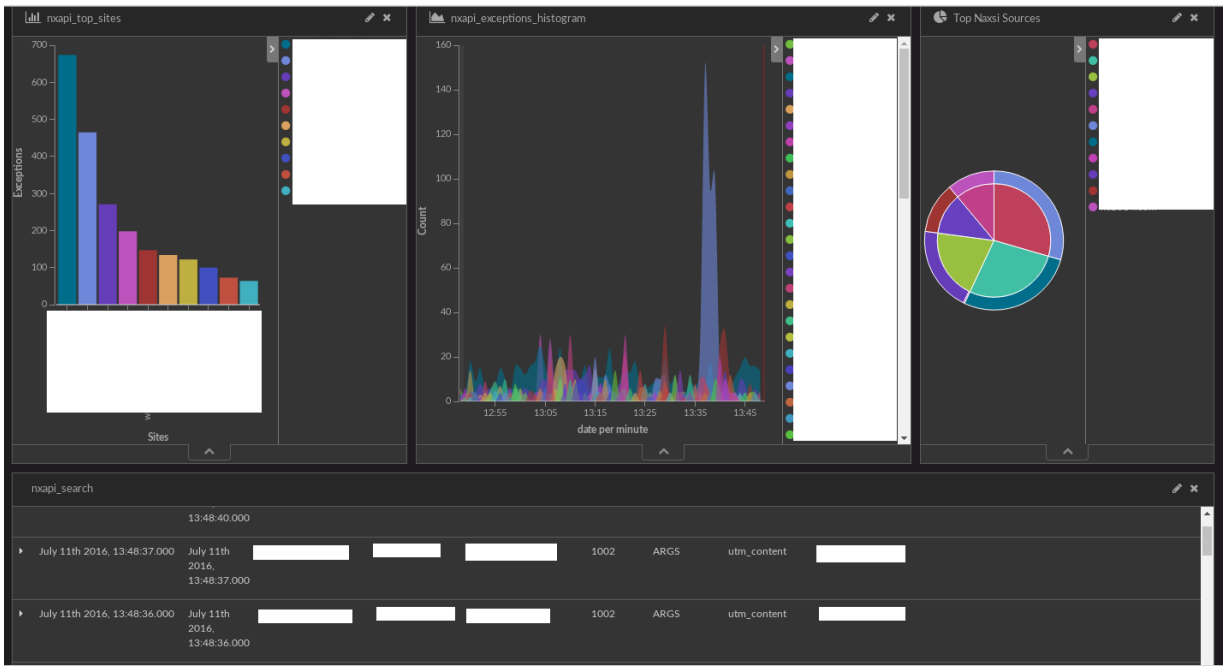


Figura 20 – Graficas en Naxsi.
https://raw.githubusercontent.com/wiki/nbs-system/naxsi/Images/kibana.png

Pero, ¿por qué es diferente?, pues contrariamente a la mayoría de los firewalls de aplicaciones web, Naxsi no se basa en una base de firmas como

un antivirus, y por lo tanto no puede ser evitado por un patrón de ataque "desconocido".

Naxsi debería ser compatible con cualquier versión *Nginx*. Depende de *libpcre* para su compatibilidad con expresiones regulares, y funciona muy bien en NetBSD, FreeBSD, OpenBSD, Debian, Ubuntu y CentOS.

Posee una amplia documentación que puede ser consultada vía web

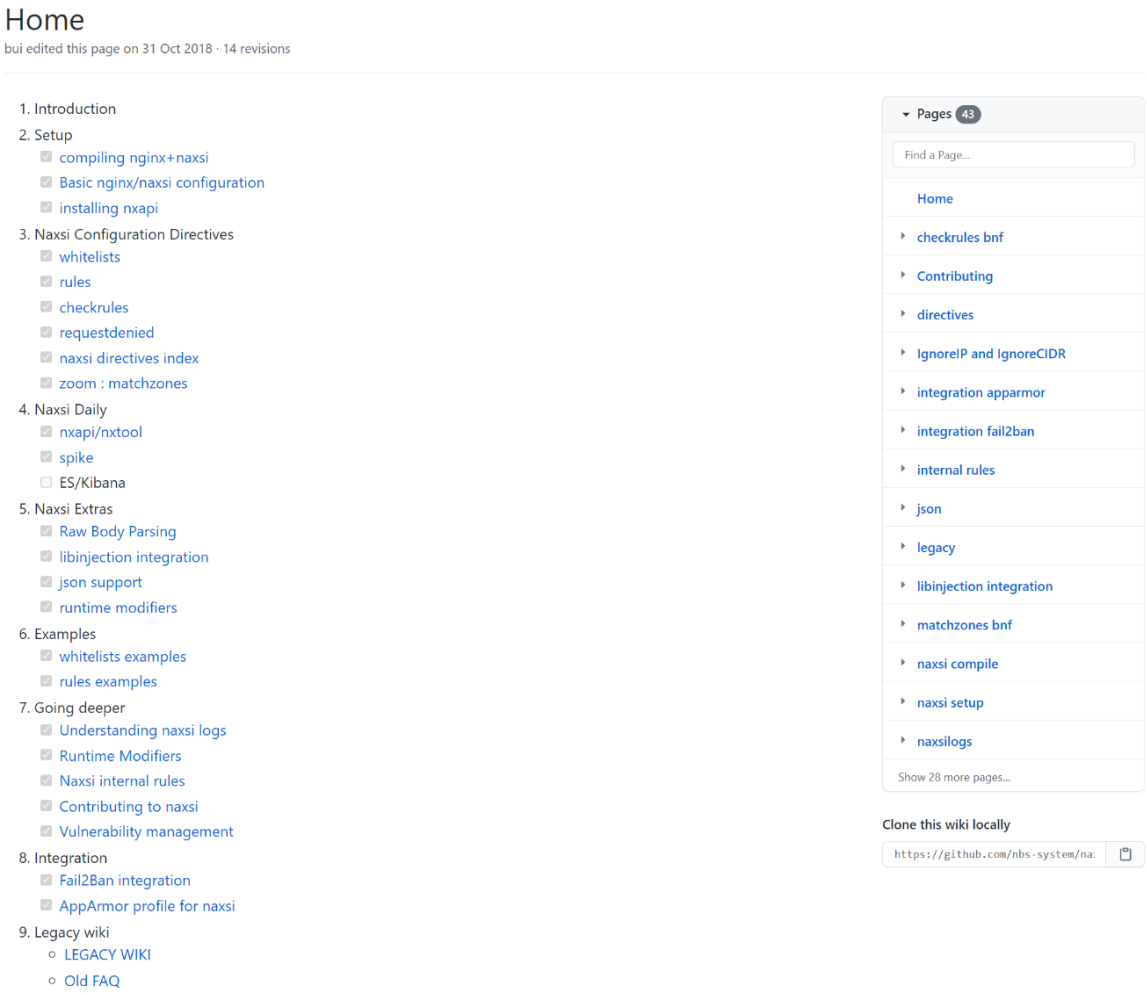
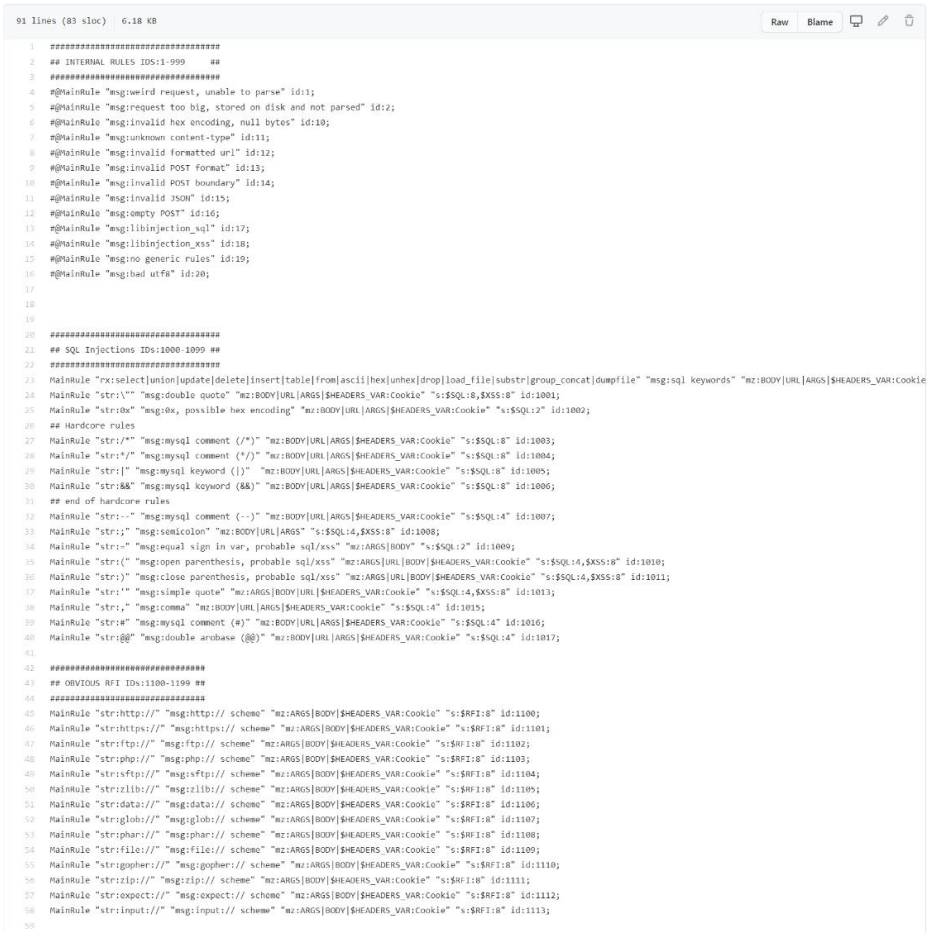


Figura 21 – Documentación de NAXSI.
https://github.com/nbs-system/naxsi/wiki

Una guía de instalación que puede ser consultada online y tutoriales que explican paso a paso como instalarlo [12]. Y también varias reglas pre establecidas que se pueden incluir para testear el WAF en la etapa inicial.

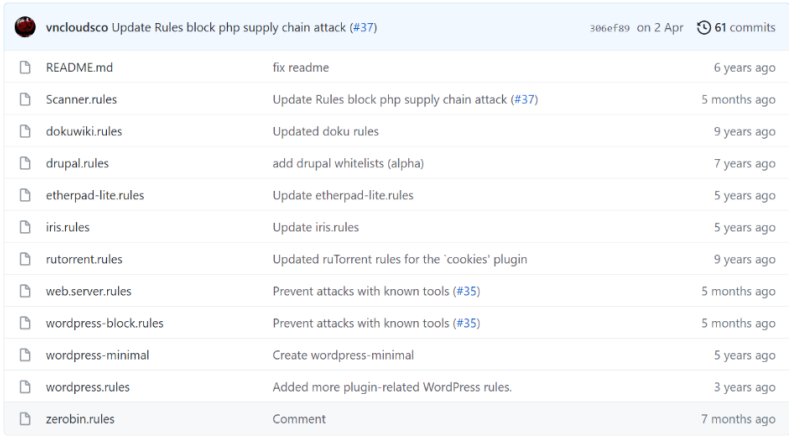


```
91 lines (83 sloc)  6.18 KB                                                    Raw  Blame

#####################################
## INTERNAL RULES IDS:1-999        ##
#####################################
#@MainRule "msg:weird request, unable to parse" id:1;
#@MainRule "msg:request too big, stored on disk and not parsed" id:2;
#@MainRule "msg:invalid hex encoding, null bytes" id:10;
#@MainRule "msg:unknown content-type" id:11;
#@MainRule "msg:invalid formatted url" id:12;
#@MainRule "msg:invalid POST format" id:13;
#@MainRule "msg:invalid POST boundary" id:14;
#@MainRule "msg:invalid JSON" id:15;
#@MainRule "msg:empty POST" id:16;
#@MainRule "msg:libinjection_sql" id:17;
#@MainRule "msg:libinjection_xss" id:18;
#@MainRule "msg:no generic rules" id:19;
#@MainRule "msg:bad utf8" id:20;



#####################################
## SQL Injections IDs:1000-1099 ##
#####################################
MainRule "rx:select|union|update|delete|insert|table|from|ascii|hex|unhex|drop|load_file|substr|group_concat|dumpfile" "msg:sql keywords" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie
MainRule "str:\"" "msg:double quote" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:8,$XSS:8" id:1001;
MainRule "str:0x" "msg:0x, possible hex encoding" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:2" id:1002;
## Hardcore rules
MainRule "str:/*" "msg:mysql comment (/*)" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:8" id:1003;
MainRule "str:*/" "msg:mysql comment (*/)" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:8" id:1004;
MainRule "str:|" "msg:mysql keyword (|)"  "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:8" id:1005;
MainRule "str:&&" "msg:mysql keyword (&&)" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:8" id:1006;
## end of hardcore rules
MainRule "str:--" "msg:mysql comment (--)" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" id:1007;
MainRule "str:;" "msg:semicolon" "mz:BODY|URL|ARGS" "s:$SQL:4,$XSS:8" id:1008;
MainRule "str:=" "msg:equal sign in var, probable sql/xss" "mz:ARGS|BODY" "s:$SQL:2" id:1009;
MainRule "str:(" "msg:open parenthesis, probable sql/xss" "mz:ARGS|URL|BODY|$HEADERS_VAR:Cookie" "s:$SQL:4,$XSS:8" id:1010;
MainRule "str:)" "msg:close parenthesis, probable sql/xss" "mz:ARGS|URL|BODY|$HEADERS_VAR:Cookie" "s:$SQL:4,$XSS:8" id:1011;
MainRule "str:'" "msg:simple quote" "mz:ARGS|BODY|URL|$HEADERS_VAR:Cookie" "s:$SQL:4,$XSS:8" id:1013;
MainRule "str:," "msg:comma" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" id:1015;
MainRule "str:#" "msg:mysql comment (#)" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" id:1016;
MainRule "str:@@" "msg:double arobase (@@)" "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" id:1017;

#####################################
## OBVIOUS RFI IDs:1100-1199 ##
#####################################
MainRule "str:http://" "msg:http:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1100;
MainRule "str:https://" "msg:https:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1101;
MainRule "str:ftp://" "msg:ftp:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1102;
MainRule "str:php://" "msg:php:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1103;
MainRule "str:sftp://" "msg:sftp:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1104;
MainRule "str:zlib://" "msg:zlib:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1105;
MainRule "str:data://" "msg:data:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1106;
MainRule "str:glob://" "msg:glob:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1107;
MainRule "str:phar://" "msg:phar:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1108;
MainRule "str:file://" "msg:file:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1109;
MainRule "str:gopher://" "msg:gopher:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1110;
MainRule "str:zip://" "msg:zip:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1111;
MainRule "str:expect://" "msg:expect:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1112;
MainRule "str:input://" "msg:input:// scheme" "mz:ARGS|BODY|$HEADERS_VAR:Cookie" "s:$RFI:8" id:1113;

```

Figura 22 – Reglas para NAXSI.

https://github.com/nbs-system/naxsi/blob/master/naxsi_config/naxsi_core.rules

Hay diferentes sets de reglas que pueden ser descargadas de GitHub e incluidas en el WAF NAXSI y que son proporcionadas y mantenidas por la comunidad de usuarios de NAXSI [13].



vncloudsco Update Rules block php supply chain attack (#37) — 306ef89 on 2 Apr — 61 commits

README.md	fix readme	6 years ago
Scanner.rules	Update Rules block php supply chain attack (#37)	5 months ago
dokuwiki.rules	Updated doku rules	9 years ago
drupal.rules	add drupal whitelists (alpha)	7 years ago
etherpad-lite.rules	Update etherpad-lite.rules	5 years ago
iris.rules	Update iris.rules	5 years ago
rutorrent.rules	Updated ruTorrent rules for the `cookies' plugin	9 years ago
web.server.rules	Prevent attacks with known tools (#35)	5 months ago
wordpress-block.rules	Prevent attacks with known tools (#35)	5 months ago
wordpress-minimal	Create wordpress-minimal	5 years ago
wordpress.rules	Added more plugin-related WordPress rules.	3 years ago
zerobin.rules	Comment	7 months ago

Fig. 23 – Set de reglas.

https://github.com/nbs-system/naxsi-rules

Naxsi es una excelente solución WAF para Ngnix, es simple, confiable y de bajo mantenimiento, a eso hay que sumarle el apoyo de la comunidad de usuarios que aportan soluciones a los inconvenientes que se puedan encontrar durante la etapa de instalación y uso de la aplicación.

Conclusión

Internet sigue creciendo continuamente y eso trae nuevas vulnerabilidades y fallas que pueden ser aprovechadas por personas malintencionadas.

Por ser los servidores más potentes que las estaciones de trabajo y estar conectados 7x24 a una dirección pública en Internet, son plataformas ideales para esconder y distribuir malware, lanzar ataques encubiertos o ser convertidos en centros de comando y control, por lo tanto, protegerlos no es sólo cuestión de cuidar los datos de la compañía y sus clientes, sino los recursos, ancho de banda y posible responsabilidad por daños a terceros.

Una aplicación web corre encima de un servidor con su Sistema Operativo, un Servidor Web (IIS, Apache, Nginx o similar), y la aplicación Web como tal, fuera de las herramientas de desarrollo como PHP, ASP, Java, Joomla u otra. Cada una de estas capas puede tener múltiples vulnerabilidades, algunas veces anunciadas, muchas veces no corregidas a tiempo. Sin hablar de las vulnerabilidades introducidas por programación defectuosa y mala parametrización de los componentes. Si la aplicación fue desarrollada por un tercero, o por nosotros mismos, tampoco tendremos la certeza de que esté libre de errores en su código o al momento de la ejecución. Una medida común de protección es un IPS [14], sin embargo, por lo general estos trabajan en capa 3 (aunque algunos poseen una variedad de protocolos que incluyen HTTP, pero no se encuentran especializados en HTTP y HTML como los *WAF*) mientras que los ataques a servidores web funcionan en capa 7 (aplicación).

La principal tarea del *WAF* es proteger las aplicaciones web inspeccionando el tráfico, sus sesiones, semántica y los ataques típicos de capa 7

sobre http/https, tales como Inyección de SQL, *Buffer Overflow*, *Cross Site Scripting*, *Cookie Poisoning*, etc.

El *WAF* tiene su capa configurable donde el administrador puede crear firmas específicas, en vez de reescribir toda la aplicación o tener que convivir con vulnerabilidades por miedo a aplicar los parches a tiempo, poner un *WAF* delante de nuestras aplicaciones nos permitirá tener una protección fuerte para esas aplicaciones sin alterar su funcionalidad o estructura. Es por ello que la implementación de complementos de seguridad como *ModSecurity*, *Shadow Daemon* o *Naxsi* se volvió muy importante.

Luego de haber presentado las tres soluciones *WAF Open Source,* queda claro que todas son buenas candidatas para ser implementadas en pequeñas y medianas empresas como así también para aquellos que tienen un emprendimiento y realizan sus negocios vía web. Las tres soluciones son similares, alguna más completa que otra, pero cualquiera de ellas puede brindarle protección contra ataques a sitios web.

Referencias

[1] OWASP German Chapter with collaboration, "Best Practices: Use of Web Application," 25 05 2008. [Online]. Available: https://owasp.org/www-pdf-archive/Best_Practices_Guide_WAF_v104.en.pdf. [Accessed 28 12 2020].

[2] Microsoft, "Microsoft documentation," 16 03 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver15. [Accessed 03 01 2021].

[3] W. Zhong, "OWASP: Command Injection," [Online]. Available: https://owasp.org/www-community/attacks/Command_Injection. [Accessed 03 01 2021].

[4] K. S, "OWASP: Cross Site Scripting (XSS)," [Online]. Available: https://owasp.org/www-community/attacks/xss/. [Accessed 03 01 2021].

[5] Nsrav, "OWASP: Denial of Service," [Online]. Available: https://owasp.org/www-community/attacks/Denial_of_Service. [Accessed 03 01 2021].

[6] "Gartner," [Online]. Available: https://www.gartner.com/en. [Accessed 20 05 2021].

[7] Ty Coon, "GNU Operating System," 1 4 1989. [Online]. Available: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html. [Accessed 15 06 2021].

[8] "github," [Online]. Available: https://github.com/zecure/shadowd_rules. [Accessed 01 08 2021].

[9] "Shadowd," [Online]. Available: https://shadowd.zecure.org/tutorials/rules/#generator. [Accessed 01 08 2021].

[10] A. TERRY, "NGINX," [Online]. Available: https://www.nginx.com/. [Accessed 19 08 2021].

[11] URI, "Wikipedia," 24 07 2021. [Online]. Available: https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme. [Accessed 19 08 2021].

[12] D. Lukan, "Protean Security, Naxsi - The Web Application Firewall for Nginx," 15 02 2014. [Online]. Available: https://www.proteansec.com/application-security/naxsi/. [Accessed 19 08 2021].

[13] "nbs-system / naxsi-rules," [Online]. Available: https://github.com/nbs-system/naxsi-rules. [Accessed 20 08 2021].

[14] "Sistema de prevención de intrusos," Wikipedia, 07 04 2020. [Online]. Available: https://es.wikipedia.org/wiki/Sistema_de_prevenci%C3%B3n_de_intrusos. [Accessed 20 08 2021].

[15] "Zecure Information Technology," [Online]. Available: https://zecure.org/. [Accessed

20 08 2021].

[16] "PHP," Wikipedia, 30 08 2021. [Online]. Available: https://es.wikipedia.org/wiki/PHP. [Accessed 02 09 2021].

[17] "Perl," Wikipedia, 22 08 2021. [Online]. Available: https://es.wikipedia.org/wiki/Perl. [Accessed 02 09 2021].

[18] "Python," Wikipedia, 28 08 2021. [Online]. Available: https://es.wikipedia.org/wiki/Python. [Accessed 02 09 2021].