Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Software

COMPARATIVA DE BASTIONADO DE UNA TIENDA ONLINE MEDIANTE WAF OPEN SOURCE

Alumno: Miguel Martín Hernando

Tutor: Julián Arroyo Álvarez

...

Agradecimientos

En este ultimo año, que finalizan mis estudios, me gustaría expresar mi más sinceros agradecimientos a todas las personas que me han acompañado y ayudado a lo largo de este trabajo y durante toda mi experiencia durante la carrera.

En primer lugar, quiero agradecer a mi tutor, Julián Arroyo Álvarez, por su apoyo, tutorías e importantes sugerencias a lo largo del desarrollo del proyecto.

Otro pilar fundamental, durante este proyecto, ha sido mi empresa (NTT Data) donde tuve la oportunidad de hacer las prácticas y aprender lenguajes y tecnologías fundamentales para poder lograr este proyecto. Gracias a todo el equipo por su paciencia y por darme un entorno tan profesional.

También quiero dar las gracias a mis profesores del Grado en Ingeniería Informática por todo su tiempo en tutorías y clases durante toda la carrera.

A mi familia, por toda su ayuda y ánimo en los momentos más difíciles. Gracias por estar siempre ahí.

Y, por supuesto, a mis amigos y compañeros de clase, con los que he compartido horas de estudio, recuperaciones, frustraciones y aprobados. Cursar la carrera con ellos ha hecho que esta experiencia sea mucho más llevadera.

¡Gracias a todos!

Resumen

Este Trabajo de Fin de Grado tiene como objetivo la comparación de distintos firewalls de código abierto (WAFs), para ver cuál se adecúa mejor a la protección de una aplicación web y al desarrollo de esta.

A lo largo del proyecto se ha realizado una investigación de qué ataques existen y un análisis comparativo entre las distintas soluciones WAF existentes en el ámbito open source (código abierto). Después, se eligió la opción que más se adecuó a los requisitos de seguridad, rendimiento y compatibilidad con la web desarrollada.

Se configurarán las reglas personalizadas del WAF, viendo su efectividad mediante ataques simulados en un entorno realista. El proyecto, en cuanto a software, también ha incluido tareas de análisis, diseño, pruebas y despliegue, demostrando la posibilidad de combinar la experiencia en software al desarrollar una aplicación web con medidas de protección o bastionado avanzadas de la misma, para lograr una aplicación segura y funcional.

Abstract

This Final Degree Project aims to compare different open-source Web Application Firewalls (WAFs) to determine which one best fits the protection and development needs of a web application.

Throughout the project, a thorough investigation into common web attacks was conducted, along with a comparative analysis of various open-source WAF solutions. Based on this analysis, the option that best met the security, performance, and compatibility requirements of the developed web application was selected.

Custom rules were also configured in the chosen WAF to evaluate its effectiveness against simulated attacks in a realistic environment. From a software perspective, the project encompassed tasks such as analysis, design, testing, and deployment, demonstrating the feasibility of combining software development experience with advanced protection and hardening techniques to create a secure and functional web application.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

En la época en la que vivimos, donde todo está digitalizado, la ciberseguridad es esencial para garantizar la seguridad y disponibilidad de los dispositivos informáticos. Hay que destacar, el aumento de amenazas y la importante necesidad de herramientas eficaces que permitan proteger tanto a empresas como a usuarios frente a posibles ataques.

Una de esas herramientas son los firewalls. Programas de software que evitan que el tráfico malicioso entre en el dispositivo que se quiera defender. Si se desea dirigirlo a aplicaciones web, se les denomina WAF (Web Application Firewall). Estos evitan ataques tales como inyecciones SQL, Cross-Site Scripting (XSS), accesos no autorizados...

En este contexto, se origina la idea de desarrollar un firewall para aplicaciones web (WAF) con el objetivo de ser el núcleo de un proyecto real. Esta idea no solo busca solucionar la necesidad de proteger las aplicaciones frente a amenazas, sino también aplicar los conocimientos aprendidos.

El diseño de este WAF se propone como solución de seguridad a una web real, uniendo la experiencia previa de desarrollo software con la ciberseguridad. Teniendo de resultado un proyecto de alto valor en el mundo real.

1.2. Motivación

Durante el tercer curso de la carrera, se tuvo la oportunidad de diseñar un firewall (no dirigido a una web, sino a un sistema operativo). Gracias a este proyecto, se entendió el objetivo y la importancia que tenían en todos los entornos presentes. Ver cómo se filtraban los paquetes según las reglas definidas, el impacto de cada petición y como prohibir determinado tráfico. Esto llevó a pensar en la posibilidad de crear un WAF en el diseño de las futuras aplicaciones.

Según se avanzó en el curso, durante el tiempo libre, el alumno se dedicó a la compra y reventa de ropa, especialmente de la marca Off-White. Se pensó la posibilidad de digitalizar el negocio. Era una idea ambiciosa, pero que se podía lograr ya que es lo que se estudió. La única pega que surgió fue, si se publica ropa en internet, deber ser de forma segura.

Gracias a lo aprendido en la carrera, especialmente en la rama de Ingeniería del Software, y en las prácticas en empresa, era perfectamente posible desarrollar una aplicación web. Y si además se combina con lo que tanto destacó en aquella asignatura optativa —la creación de un firewall—, se formó, la oportunidad de realizar un TFG interesante, práctico y completo, en el que también se pudiera investigar a fondo los distintos tipos de ataques web y cómo prevenirlos.

Este enfoque permitió llenar un vacío que se siente en la carrera, más en particular en la rama de ingeniería de software: si bien aprendes a diseñar y crear software eficiente, siempre queda la curiosidad de crear software seguro y preparado para enfrentarse al mundo real.

En definitiva, este proyecto surge de la combinación entre lo aprendido en la rama de software, objetivos personales y la necesidad real de proteger aplicaciones web frente a amenazas cada vez más frecuentes. Desarrollar un WAF para un negocio digitalizado no solo es un reto académico, sino también una oportunidad para aplicar de forma práctica conocimientos y dar una solución de seguridad en un entorno real.

1.3. Objetivos

Objetivo general

El objetivo principal de este proyecto de fin de grado es desarrollar y diseñar un WAF (Web Application Firewall) propio, que pueda detectar y detener ataques comunes que puedan poner en riesgo la aplicación web creada —aplicación web de compraventa de ropa—.

Se trata de crear un proyecto personal real seguro. Tanto para el desarrollador que gestiona los datos como para los usuarios que accedan a él. Este trabajo de fin de grado no solo busca usar lo aprendido durante la carrera, más en especial en la rama de software y en las prácticas de empresa, sino también realizar una investigación en profundidad de un aspecto que, en opinión general, muchas veces, queda un poco de lado a la hora de desarrollar software, la seguridad.

De esta forma, el trabajo de fin de grado tiene un enfoque formativo doble: por un lado, desarrollar una aplicación web funcional y, por otro lado, diseñar mecanismos de defensa mediante un firewall para defender dicha web frente a ataques y amenazas.

Objetivos específicos

Estudiar e Investigar qué ataques existen y cuáles son los más comunes. Ver cómo afectan, su gravedad y cómo se pueden evitar (inyecciones SQL, XSS, accesos no autorizados, ataque de fuerza bruta y otros elementos relacionados con manipular el tráfico de http).

Analizar e investigar qué tipos de WAF existen, cómo funcionan, ver qué limitaciones ofrecen, qué técnicas usan frente a tráfico malicioso y elegir cuál de los existentes es el que mejor se adapta a la web y los requisitos que queramos lograr.

Diseñar y crear la web con el lenguaje aprendido en el periodo de prácticas junto a tecnologías también vista ahí.

Integrar una pasarela de pagos o un entorno de pagos para desarrolladores, para simular un entorno realista y seguro para el usuario.

Una vez elegido el WAF y la web ya diseñada, habrá que diseñar y desarrollar sus reglas, configuración y metodologías, adaptado a las necesidades de la web sin que afecte negativamente a su rendimiento.

Poner a prueba la web con ataques simulados, evaluando su efectividad para detectar y bloquear amenazas reales.

Configurar un dominio accesible desde fuera de localhost, ya sea a través del nombre de host de la propia máquina o mediante túneles. Se busca así lograr un entorno más realista y permitir el acceso desde otros dispositivos para pruebas y demostraciones.

1.4. Estructura de la memoria

Este documento se estructurará de la siguiente manera:

Capítulo 2 Metodología: Se describe el modelo de proceso y la metodología de desarrollo adoptada. Se identifican que partes interesadas hay implicados y se detallan sus requisitos mediante un product backlog inicial.

Se analizan los riesgos del proyecto y la planificación temporal a través de un cronograma. Finalmente, se realiza una estimación de los costes y recursos necesarios.

Revisa la evolución del proyecto respecto a la planificación inicial, comentando los posibles desvíos, herramientas de gestión utilizadas y decisiones tomadas en el camino. También se evalúan los costes estimados.

Capítulo 3 Tecnologías utilizadas: Presenta las diferentes herramientas y entornos que se han empleado durante el desarrollo de este proyecto.

Capítulo 4 Análisis: Detalla en profundidad que funcionalidades y requisitos se necesitan implementar antes de desarrollar la web. Comprender qué se quiere resolver.

Capítulo 5 Diseño: Convierte el análisis en una solución técnica, determinando la arquitectura y cómo resolverá el sistema los requisitos de análisis.

Capítulo 6 Seguridad y comparativa: Capítulo dedicado al estudio de la seguridad en aplicaciones web. Se analizan los principales tipos de ataques y se realiza una comparativa entre distintas soluciones WAF de código abierto. Se justifica la elección de una de ellas en base a su efectividad, rendimiento y facilidad de integración.

Capítulo 7 Prueba practica de seguridad: Describe las tareas hechas durante el desarrollo de la aplicación web y la configuración del WAF. Incluye pruebas funcionales y de seguridad con ataques simulados.

Capítulo 8 Conclusiones: Resume los resultados obtenidos, reflexiona sobre el cumplimiento de los objetivos planteados y presenta posibles líneas de mejora o trabajo futuro.

Capítulo 2

Metodología

En este capitulo se explicará la metodología que se ha usado, junto con los requisitos y planificación esperadas para el proyecto, es decir, la estimación que se espera que se va a lograr. Finalmente se mostrará el seguimiento que ha abordado el proyecto y su control con lo estimado:

2.1. Requisitos y planificación

Para la gestión del proyecto se optó por una metodología ágil basada en Scrum, permitiendo un trabajo incremental. Scrum permite dividir el proyecto en etapas cortas llamadas sprints, de forma que el proyecto se pueda adaptar de la mejor forma posible a cambios o resultados imprevistos. La definición oficial de scrum la define como un 'un marco ligero que ayuda a equipos, o en este caso al proyecto, a dar un valor a través de soluciones que se adaptan a problemas complejos.

Cuando se trabaja en scrum se establecen roles y eventos bien diferenciados. El Scrum Master da un entorno corporativo y el Product Owner (Propietario del producto) ordena y prioriza el Product Backlog. En cada sprint se seleccionan los elementos del Backlog que los miembros del equipo convierten en un incremento del producto. Al final de cada sprint, el equipo junto a las partes interesadas (stakeholders) para ver los resultados y ajustan el plan para el siguiente ciclo. Este enfoque iterativo era perfecto para comparar varios WAF paso a paso en cada ciclo.

Stakeholders

En este proyecto, se crea un producto para usuarios directos que lo vayan a usar o partes interesadas. Se pueden identificar como:

- Tutor académico: Actúa como cliente interno que define requisitos de seguridad y funcionalidades del sistema, revisa entregables y los valida.
- Administrador del sistema (Estudiante): Mantiene y despliega el entorno (Contenedores Docker).
- Equipo de desarrollo (Estudiante): Responsable de probar y documentar las comparativas entre los WAFs. Como parte interesada, es afectada por planificación y recursos disponibles.

Product Backlog

El backlog inicial que se creó, estaba compuesto por varias épicas. reflejando las necesidades de los stakeholders a alto nivel. Se ha seguido una plantilla para definir las historias del usuario:

- Como administrador quiero proteger la web contra ataques para garantizar la integridad de los datos, documentando y registrando los ataques.
- Como desarrollador, quiero comparar distintos WAF (BunkerWeb, Coraza y ModSecurity) para elegir el que mejor rendimiento dé. Para luego disponer de un entorno de contenedores para instalarlo.
- Como coordinador del proyecto quiero gestionar las versiones de gitLab y el código de la aplication web.
- Como usuario final quiero que las transacciones con PayPal se hagan de forma segura.

cada épica sirve de base para dividir el trabajo en historias de usuario y tareas mas en detalle para sprint posteriores. Este enfoque de scrum, garantiza que cada requisito se centre en el valor entregado al stakeholder correcto.

Análisis de riesgos

Se realizó un análisis para cuantificar los riesgos e identificar posibles amenazas durante el proyecto. Existen diversas categorías de riesgos que pueden retrasar y afectar al proyecto. Los riesgos comunes tienen factores relacionados con presupuestos, recursos y el cronograma. Resumen de los principales riesgos previstos:

Riesgo	**Categoría**	**Descripción breve**
Retrasos	Cronograma	La evaluación de cada WAF podría demorar más de lo previsto (problemas técnicos con BunkerWeb/Coraza). Esto impactaría la fecha de entrega.
Limitación	Recursos humanos	El estudiante es único desarrollador; carece de experiencia en WAFs, por lo que requerirá curva de aprendizaje.
Problemas de compatibilidad	Técnico	Dificultades al integrar un WAF con Apache+PHP/Docker (por ejemplo, configuraciones complejas o dependencias faltantes).
Fallos de seguridad no detectados	Técnico	Un WAF mal configurado podría no mitigar ciertos ataques (inyección SQL, XSS, etc.). Riesgo de vulnerabilidades residuales.
Incremento de costes	Financiero	Posible necesidad de licencias, servicios adicionales o hardware (aunque se planea usar software libre). El presupuesto podría aumentar.
Cambios en requisitos	Alcance	El tutor u otro stakeholder podría solicitar cambios en funcionalidades de seguridad a mitad del proyecto, requiriendo replanificación.
Fallas de terceros (PayPal Sandbox)	Externo	Dependencia de servicios externos (PayPal Sandbox, proveedores de túneles como LocalTunnel). Suspensiones o cambios en estos servicios retrasarían pruebas.

Tabla 2.1: Identificación de riesgos del proyecto

Los riesgos de categoría 'Técnico' se centran en la protección de datos y mantener el software, lo cual es importante en un proyecto de seguridad. Es por eso que se identifican estos riesgos. Para cada riesgo se planeta una mitigación, para tener una capacidad de arreglo en caso de producirse. Se asignó prioridad alta a los riesgos de cronograma, debido a que un retraso compromete todos los objetivos del TFG.

Planificación del cronograma

La planificación que se tuvo al inicio, tuvo en cuenta 3 meses de marzo a mayo, distribuido en varios sprints divididos por quincenas. Cronograma:

- Marzo (Sprint 1): inicio del proyecto. Se trasteó con Juice shop, se vieron vulnerabilidades y se empezó a estudiar sobre seguridad. Se configuró en entorno, instalando Docker Desktop, Visual Studio Code y creación del repositorio en GitLab, definiendo el Backlog inicial. Se hicieron reuniones de planificación con el tutor para detallar requisitos. La meta de este sprint fue dejar preparado el entorno de pruebas.
- Abril (Sprints 2-3): Pruebas con BunkerWeb. En el Sprint 2 se instaló este WAF en docker. Se documentaron resultados. En el Sprint 3, se repitieron pruebas para refinar la configuración. Paralelamente se inició una evaluación de Coraza, en las ultimas semanas de abril, para ver su funcionamiento. Al finalizar abril, ya tenían pruebas básicas de estos 2 WAFs.
- Mayo (Sprints 4-5): Evaluación de ModSecurity. En el Sprint 4 se completó la comparación de los 3 y se decidió instalar ModSecurity en el entorno junto a PHP con Apache. Se diseñó y creó la web, seguido de las pruebas funcionales. En el sprint 5, se terminó de configurar el WAF y se dieron los últimos retoques a la web. Finalmente solo quedaba hacer la memoria.

En este plan se consideraba que 3 meses serian suficientes. Pero, en la practica surgieron dificultades técnicas que exigieron una pequeña extensión hasta junio. Principalmente, la evaluación de BunkerWeb y en parte Coraza, llevó mas tiempo de lo previsto (Poca documentación y errores), de manera que se tuvo que añadir una semana extra de pruebas.

Presupuesto

El proyecto se basa principalmente en software libre, por lo que el presupuesto económico será mínimo. Pero, es estima los siguientes costes, que no necesariamente son siempre monetarios:

- Recursos humanos: El estudiante dedicará 20 horas semanales, durante 4 meses. Aunque no conlleva coste salarial (Trabajo académico), inversión en tiempo.
- Infraestructura: Se usarán dos ordenadores, para trabajar en cualquier lugar. Un portátil para trabajar en la universidad y otro de sobremesa para trabajar desde casa. También es necesaria una conexión a internet.
- Software: Visual Studio, Docker Desktop, GitLab, php, apache, ModSecurity, PayPal sandbox, LocalTunnel... Son software libres sin licencia, aquí no hay gastos ni se prevén licencias.
- Otros servicios: Se consideraron Ngrok o Cloudflare en vez de LocalTunnel, pero se optó mejor por una opción gratuita.

- Gastos menores: Electricidad e internet durante todo el proyecto (20€), impresión del documento (20€) y gastos en transporte (10€).

En total, el presupuesto total estimado es de 50€ mas o menos, dado el uso de recursos de software libre. La parte mas importante fue el tiempo y su gestión para los sprints.

2.2. Seguimiento del proyecto

Aquí abordaremos lo que el proyecto realmente ha ejecutado y que control ha tenido durante el tiempo de desarrollo.

Ejecución real del proyecto

En la fase de ejecución se siguió fundamentalmente el plan establecido, con las siguientes observaciones de lo realmente realizado:

- Se creó el entorno de desarrollo con Docker Desktop tal como estaba planificado. Se levantaron contenedores separados: uno para la base de datos (MySQL) y otro para Apache+PHP con el WAF en pruebas. Composer se utilizó para gestionar dependencias de PHP necesarias relacionadas con la pasarela de pagos. GitLab se configuró con ramas para cada sprint y se documentaron los cambios en commits regulares.
- Se realizaron las pruebas con PayPal Sandbox integradas y se configuró LocalTunnel para exponer localmente la URL (se descartó usar ngrok o Cloudflare debido a restricciones de tiempo y configurabilidad). Este entorno permitió simular transacciones reales durante las evaluaciones de WAF.
- En cuanto a los WAFs: BunkerWeb no se instaló tan satisfactoriamente y surgieron errores al aplicar ciertas reglas por falta de documentación y sobre todo en español. Coraza también no dio un resultado óptimo. Se documentaron exhaustivamente ambos procesos y su rendimiento. Finalmente, se implementó ModSecurity en Apache (usando el módulo libmodsecurity) y se importó el OWASP. Se ensayaron configuraciones de ejemplo para bloquear inyecciones SQL, XSS, etc.
- Cada sprint terminó con una pequeña demo al tutor. Los entregables incluían un reporte de pruebas por WAF y el código en GitLab. Las revisiones diarias fueron breves, dado que el equipo estaba formado por una sola persona, pero se revisó el estado de tareas en GitLab.

Desviación respecto al plan inicial

Durante el proyecto, se formaron pequeñas desviaciones como:

- Extensión del plazo: como se había previsto en el cronograma actualizado, el proyecto se extendió hasta junio. La causa principal fue la dificultad técnica al evaluar BunkerWeb y Coraza: se retrasó una semana las pruebas iniciales. Esto afectó a la planificación de los últimos sprints.
- Repriorizar sprints: debido a esos retrasos, se ajustó el alcance de los sprints 4 y 5, posponiendo parte de la documentación final al sprint de junio. Se priorizó completar pruebas sobre la documentación, para asegurar datos comparativos válidos.
- Uso de herramientas de gestión: todo el seguimiento se realizó en GitLab. Simplificó la gestión centralizando código en una sola plataforma, permitiendo trabajar desde los dos dispositivos disponibles.
- Metodología: se mantuvo Scrum y no se cambió a otra metodología, pero al ser un solo desarrollador, algunas partes del proyecto fueron informales (stakeholders). Se respetó el enfoque ágil y se adaptó al proyecto individual.

En general, aunque hubo retrasos, estos se gestionaron internamente ajustándose al sprint adicional sin comprometer el alcance final (se logró comparar todos los WAFs y producir la comparación). Las desviaciones del cronograma fueron comunicadas oportunamente al tutor, quien aprobó la ampliación.

herramientas usadas para la gestión

El uso de herramientas no ha cambiado de como se planificó. Las principales herramientas de soporte y gestión empleadas finalmente fueron:

- GitLab: fue el repositorio de código y el gestor de versiones. Se creó una rama para cada sprint y Issues para cada tarea en el backlog. De esta forma, el plan de proyecto quedó reflejado en GitLab: el equipo coge tareas del backlog en cada sprint.
- Visual Studio Code: IDE de desarrollo para editar código PHP, reglas del WAF y documentación. Incluyó extensiones para Git y Docker. No se invirtió en licencias de IDEs comerciales.
- Docker Desktop: permitió optimizar todo el entorno en contenedores. Se creó el Docker Compose con los servicios (Apache+WAF+php y BD) y el Dockerfile para la imagen del contenedor del WAF, facilitando la replicación del entorno.
- PayPal sandbox con composer: plataforma de PayPal para simular pagos reales sin transacciones monetarias. Se utilizó para validar la integración con la web.
- LocalTunnel: servicio para exponer localmente un puerto con un dominio público. Fue útil para recibir usuarios desde otro ordenador y que la experiencia de uisuario sea mas real, sustituyendo opciones como ngrok.

- Documentación Látex: El informe se redactó en LaTeX y se usaron plataformas de mensajería (correo electrónico) para seguimiento. No se utilizó software adicional de gestión de proyectos aparte de lo mencionado.

En conjunto, la elección de estas herramientas (todas gratuitas) permitió centralizar el trabajo. Por ejemplo, los Commits frecuentes reflejaron el avance real del proyecto, facilitando informes de seguimiento.

Evaluación de riesgos reales

Durante la ejecución del proyecto, surjieron varios riesgos identificados que se manifestaron de forma considerable:

- Retrasos en el cronograma: efectivamente, los problemas de configuración de BunkerWeb y Coraza impactaron el cronograma. Para gestionarlo, se replanificó el sprint extra de junio y se priorizaron objetivos críticos. Este riesgo se mitigó anticipando más horas de pruebas en docker.
- Falta de experiencia: al inicio, la curva de aprendizaje de nuevos WAF fue muy elevada. Se abordó mediante autoformación, lectura de documentación (inglesa mayormente). Ya que en cuanto al desarrollo web no hubo problemas y todos los plazos fueron cumplidos. Debido a que ya se tenían los conocimientos previos para lograr un software eficiente y funcional.
- Compatibilidad: surgieron errores al integrar WAF con Apache en contenedores y en docker, sobre todo en BunkerWeb y Coraza. Se gestionaron instalando versiones de software compatibles recomendadas.
- Seguridad real: para asegurarse de que las configuraciones del WAF bloqueaban efectivamente ataques, se corrieron pruebas manuales. En ningún caso real se detectaron brechas serias durante las pruebas finales, lo que indica que el riesgo fue controlado.
- Problemas con servicios externos (externo): el uso de PayPal Sandbox y LocalTunnel resultó estable.

En cada sprint se revisaron los riesgos, evaluando su probabilidad e impacto real. Para los riesgos que se hicieron reales, se planificaron acciones de mitigación: reasignar tiempo, buscar ayuda y ajustar el backlog. En general, la gestión de riesgos fue efectiva, ya que ningún imprevisto bloqueó el proyecto, solo generó algún retraso.

Costes reales

Los costes reales del proyecto quedaron prácticamente iguales que con las estimaciones iniciales. No se pagaron licencias ni servicios adicionales fuera de lo previsto. Todas los herramientas usadas fueron gratis y disponibles.

El único gasto económico real registrado fueron conceptos menores (electricidad, impresiones, transporte) por valor inferior a 50€. El "coste" en horas de trabajo fue el tiempo total del estudiante (360 horas en cuatro meses aproximadamente), que coincide con la estimación de carga laboral. No hubo horas extra sustanciales fuera del plan de sprints.

En consecuencia, no hubo cambio en el presupuesto. El presupuesto de tiempo y recursos se respetó gracias al scrum seguido: al aumentar un poco el plazo, se completó todo sin reducir el alcance. El resultado fue la elección de ModSecurity como WAF definitivo, cumpliendo los objetivos de seguridad y funcionalidad planeados dentro de los recursos estimados.

Capítulo 3

Tecnologías utilizadas

3.1. Docker

Para desplegar la aplicación web, ha resultado más fácil utilizar Docker como entorno principal de ejecución. Había otras alternativas como usar máquinas virtuales o instalar directamente un host físico, pero resultó más cómodo utilizar Docker y más en especial su aplicación de escritorio.

Docker desktop hace sencillo tanto crear contenedores como ejecutar entornos. Sobre todo, porque ya se enseñó en DBCS (Desarrollo basado en sistemas y servicios) y además es muy usado en entornos laborales reales como en las prácticas de empresa.

Es cierto que el sistema operativo ideal para usar Docker es Linux, pero gracias a Docker desktop (que también es posible instalarlo en algunas distribuciones de Linux) hace que Docker sea muy fácil de usar también en Windows 11 como en el caso de este proyecto.

Docker permite introducir servicios dentro de los llamados contenedores, los cuales son ligeros, portables y fácilmente replicables. A diferencia de una máquina virtual que necesita un sistema operativo y muchísimos recursos, los contenedores Docker se ejecutan en el mismo sistema operativo en donde están instalados y solo necesitan una imagen en la que basarse para funcionar de forma rápida y eficiente. Esto unido a la interfaz intuitiva que ofrece Docker desktop, se puede lograr un entorno bien estructurado en apenas segundos.

Además, Docker ofrece una capa aislada entre los diferentes servicios o contenedores que hay comunicándose que da bastante seguridad al trabajar con bases de datos o servidores webs expuestos como es este caso.

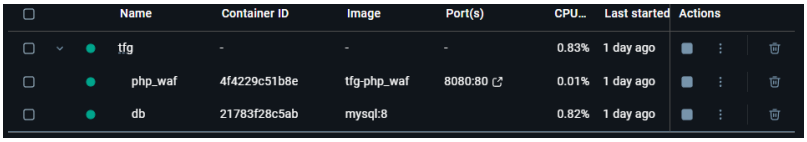

Figura 3.1: Arquitectura de contenedores con Docker

Como se puede ver en la imagen, se ha optado por diseñar el TFG con solo dos contenedores. Cada uno de ellos con su nombre (Aquí puedes poner el que se quiera), su ID (lo asigna el propio Docker al crear los contenedores), la imagen usada (la imagen en la que se basa cada contenedor, en este caso el contenedor de la base de datos se base en una imagen de mysql:8 y el otro en una imagen creada desde el Dockerfile), el puerto donde se va a exponer ese servicio (Este se puede configurar desde el docker-compose.yml, no se expone la base de datos a ningún puerto por motivos de seguridad, así solo accede el otro contenedor) y por último muestra el porcentaje de CPU que necesita para ejecutar el contenedor.

Todas estas características se pueden definir dentro del archivo docker-compose.yml:

```
Run All Services
services:

  Run Service
  db:
    image: mysql:8
    container_name: db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: melvin
      MYSQL_DATABASE: TFG_db
      MYSQL_USER: melvin
      MYSQL_PASSWORD: melvin
    volumes:
      - db_data:/var/lib/mysql
```

Figura 3.2: Contenedor db

En el contenedor "`db`", se puede definir: la imagen, el nombre, las credenciales (para acceder a la base de datos y configurarla, ya que al no tener un puerto expuesto al exterior se tiene que configurar desde dentro) y por último el volumen donde se guarda toda la información, los usuarios, pedidos, etc. Ya que si no se pone un volumen cada vez que se bajen los contenedores se perdera esa información.

Figura 3.3: Contenedor php_waf

En cuanto al contenedor "php_waf", no tiene imagen, ya que se 'construye' (build) desde el Dockerfile. Tiene definido el nombre y el puerto donde ese contenedor estará escuchando para acceder a él.
También tiene una dependencia a la base de datos para que se pueda comunicar con ella y un volumen para mantener la misma información que hay en la máquina local, al contenedor.
Ya que copia toda la raíz del proyecto (./) en la ruta del contenedor (/var/www/html). Teniendo los mismos datos en el entorno local que en el contenedor. Al final el contenedor actúa como una máquina virtual independiente y necesita que se le 'copie' la información.

Este paso de información se podría hacer directamente desde el Dockerfile, copiando lo que hay en una ruta en la otra, pero hay una pega, si se hace desde el Dockerfile cada vez que se hace un cambio en el proyecto se tendrían que bajar y subir los contenedores para observarlo y eso llevaría mucho tiempo. Para ello se deja en el docker-compose para más comodidad.

Ocurre algo parecido con el puerto, el contenedor al actuar como una máquina virtual tiene sus propios puertos. Entonces deberá "ligar" el puerto del ordenador local al del contenedor. De ahí esa sintaxis tan característica de Docker al definir los puertos (8080:80). 8080 es el puerto del ordenador local y el 80 el de Docker.

En cuanto a la información y configuración del WAF, dentro del contenedor, se prefirió hacerlo desde el Dockerfile ya que es desde donde se clona el repositorio de Modsecurity.

Es decir, al igual que en la web, hay que copiar los archivos del entorno al contenedor, en el WAF se hace lo mismo. Se han creado los archivos necesarios para el entorno y desde el Dockerfile se copian al contenedor. El Dockerfile se explicará en el capitulo de seguridad, ya que explica como instalar el WAF y tiene mas sentido desarrollarlo en ese capitulo.

Aunque lo habitual y quizá más lógico sería hacer un contenedor por servicio, es decir, hacer 3 contenedores:

- Modsecurity + Apache
- PHP + Apache
- base de datos

Se ha optado por hacer solo 2 con la finalidad de hacerlo mucho más robusto y evitar problemas de comunicaciones entre mas contenedores.

- base de datos (Sigue estando en un contenedor aparte).

- PHP + Apache + Modsecurity (Se combinan los otros dos en uno más seguro).

De esta forma al tener en el mismo contenedor PHP + Apache + Modsecurity se logra un único punto de entrada seguro y protegido por el firewall. Creando un Dockerfile más limpio y lógico que si hubiera más contenedores.

Además de que a la hora de configurar el WAF, sus reglas, configuraciones, nombre de dominio para el puerto expuesto. Se hacía mucho más fácil con un solo contenedor (PHP + Apache + Modsecurity).

También a la hora de reducir la velocidad y ahorrar memoria de CPU. Es mucho más efectivo un solo contenedor individual (sin contar con la base de datos) que tener dos comunicándose, dando lugar a error y una menor efectividad. Por motivos ya vividos tener 3 contenedores al final es mucho menos escalable que tener solo dos.

En resumen, con 2 contenedores se gana más robustez, tanto a la hora del Dockerfile, como al hacer el docker-compose. Teniendo en el Dockerfile configurado todo el WAF y en el docker-compose los volúmenes con los datos de la web. Se puede hacer de las dos formas. Pero siempre hay que pasarle la información necesaria de tu entorno al contenedor.

3.2. Servidor Web

Un servidor web es un componente esencial en la arquitectura de este proyecto. Es el encargado de hacer funcionar correctamente el firewall y la web. Ya que sin un servidor donde estos dos componentes se ejecuten no podría ser posible este proyecto.

Un servidor web se ejecuta en el propio ordenador con la función de guardar y procesar las peticiones que los usuarios generan a través de navegadores (lo más común) a por ejemplo una web. Este proceso se hace mediante el protocolo HTTP (hyper text transfer protocol) o en su defecto su versión segura certificada HTTPS.

Estas peticiones se hacen una URL, por ejemplo "offmyle.loca.lt", enviando una solicitud HTTP a la web donde se aloja esa URL. El servidor la recibe y devuelve el recurso solicitado, normalmente una pagina HTML, un archivo, etc.

Estos recursos devueltos pueden ser de dos tipos, dinámicos y estáticos. Estáticos son aquellos recursos que no cambian como HTML, CSS, imágenes, etc. El servidor simplemente lo envía sin modificarlos.

Pero los recursos estáticos se generan en tiempo real según la petición. Por ejemplo, Iniciar sesión. Es por ello que lenguajes como PHP funcionan muy bien en el lado del servidor ya que permiten una experiencia dinámica al usuario.

Estos conceptos son importantes de entender ya que en este TFG hay una web y en su defecto habrá que aplicar todos estos conceptos.

En el desarrollo de este proyecto, un elemento clave fue elegir con que servidor funcionará la web y el cortafuegos de esta (WAF). Tras analizar diferentes alternativas, finalmente se optó por Apache HTTP server (comúnmente conocido como Apache) como servidor principal.

Esta decisión tuvo su sentido, ya que hubo una investigación sobre las consideraciones técnicas que hacían de Apache la opción más adecuada para los objetivos del proyecto. Pero antes de meternos en Apache y explicar por que es la mejor opción. Se procederá a explicar, que alternativas hubo hasta llegar a él. Por ejemplo, Nginx, fue la alternativa que en un principio se tomó como servidor principal del proyecto pero se acabó descartando.

Durante la fase de diseño inicial se evaluó Nginx como posible servidor web alternativo. Nginx es ampliamente reconocido debido a su rendimiento y eficiencia, siendo bastante efectivo en entornos con alto tráfico concurrente (Tiene una arquitectura basada en eventos).

No obstante, al tratarse de un proyecto que prioriza la seguridad, la integración nativa con PHP y la facilidad de configuración de WAF, Nginx no ofrecía tanta compatibilidad con las tecnologías que querían usar. El mayor desafío a la hora de usar Nginx fue su integración con Modsecurity.

Aunque existen versiones de Modsecurity para Nginx (como Modsecurity v3), estas implementaciones no ofrecen el mismo grado de estabilidad ni la misma cantidad de características disponibles que si usamos Apache.

Por ejemplo, existen determinadas directivas avanzadas que no estaban completamente soportadas en la versión de Nginx, y ciertas configuraciones que funcionan correctamente en Apache presentaban limitaciones o necesitaban modificaciones a mayores para poder adaptarse a Nginx.

Esto afectaba directamente al desarrollo de reglas personalizadas de seguridad, que en un elemento clave en este proyecto y también a las reglas por defecto que tiene Modsecurity.

En cuanto a PHP, la ejecución en Nginx requiere una configuración adicional mediante FastCGI (habitualmente usando PHP-FPM), lo cual puede complicar ligeramente la arquitectura de despliegue y la resolución de errores.

Es por eso, que en las prácticas de empresa al utilizar PHP con Nginx, se valoró implementarlo en el proyecto. Pero a la hora de hacerlo, surgieron diversos inconvenientes en cuanto a configurarlo, se descartó ya que Apache no daba tantos problemas ni requería tanta configuración.

Aunque utilizar Nginx es completamente válido, solo era eficiente en proyectos profesionales a gran escala, para un proyecto de tipo académico donde solo ha habido una persona desarrollando el proyecto, se busca más control sobre cada componente y una integración sencilla entre servicios utilizando Apache, siendo una opción más accesible y directa.

Apache es uno de los servidores web más utilizados a nivel mundial desde hace décadas. Tiene una amplia documentación, siendo una herramienta muy robusta para proyectos tanto personales como profesionales.

Una de las principales razones para elegir Apache en este proyecto ha sido su excelente compatibilidad con el lenguaje PHP, que es el lenguaje elegido para el desarrollo de la aplicación web y también porque es el servidor más cercano que se ha tenido durante la carrera.

Apache permite la integración directa de PHP (FROM php:8.2-apache) mediante ese módulo, que facilita una ejecución rápida y sencilla sin necesidad de intermediarios adicionales. Al ser nativo, simplifica el despliegue, la depuración de errores y el mantenimiento general de la aplicación.

Además de PHP, otra pieza esencial del proyecto es Modsecurity. Cuando se creó fue diseñado para funcionar de forma nativa con Apache, y aunque hoy en día existen métodos para implementar otros servidores como Nginx, la integración con Apache sigue siendo la más estable.

En Apache, Modsecurity se ejecuta como un módulo integrado (libapache2-mod-security2 visto en la parte de Docker), lo que permite una configuración directa dentro del servidor y de las reglas de seguridad.

Pero el factor que influenció la decisión de elegir Apache fue la experiencia adquirida durante la carrera. Ya que fue el servidor web utilizado en varias asignaturas relacionadas con la administración de sistemas y desarrollo web, por lo que su configuración y estructura ya eran algo conocidos.

Esta experiencia no solo permitió acelerar el desarrollo inicial del entorno, sino que también facilitó la integración del WAF y la depuración de errores durante las pruebas de seguridad. Si bien en entornos de prácticas empresariales se había utilizado Nginx, su adaptación a un entorno personal presentaba algunas incompatibilidades específicas con ciertas versiones, reglas o herramientas utilizadas (sobre todo a la hora de crear los contenedores).

Debido a estos problemas en Docker, llevó a reconsiderar su uso buscando una solución más estable y fiable como Apache (Servidor que aloja tanto el WAF como de la web, necesitamos que sea seguro y estable), cuya integración con Modsecurity fue mucho más testeada, especialmente dentro de entornos basados en Docker, como este proyecto.

3.3. Entorno de desarrollo (IDE)

En cuanto al desarrollo de este proyecto o en cualquier desarrollo en general, el entorno de desarrollo integrado tiene un papel fundamental a la hora de trabajar de forma cómoda y organizada.

No se trata únicamente de escribir código, ya que para ello no es totalmente obligatorio un entorno de desarrollo integrado, sino de tener un entorno que facilite tanto desarrollar pruebas, controlar versiones, estructurar archivos y, en consecuencia, tener todo el entorno bien estructurado.

Si se trabaja sin un entorno integrado, la corrección de código, pruebas y compilación se hace menos productiva. En un entorno integrado la depuración es mucho mas efectiva y puede dar al instante consejos sobre errores o posibilidad de arreglar fallos.

También si se quiere trabajar con un control de versiones y repositorios git, los hace esenciales a la hora de trabajar en grupo (creando ramas) e incluso en solitario. Debido a que se puede usar git como una nube de código y guardar ahí proyectos o archivos creados desde distintos dispositivos. Los ides incluyen una interfaz bastante fácil de usar (sin tener que usar comandos) en cuanto al uso de git.

Durante la carrera, incluso nada mas se empieza, en algunas asignaturas de primero, ya hay contacto con entornos de desarrollo integrado y se tiene la posibilidad de usar git. El primero fue eclipse cono fundamentos de programación. Y más adelante se descubrieron otros como Pycharm, Netbeans, SublimeText (Este no es un entorno de desarrollo integrado como tal, pero para desarrolladores es de gran ayuda), entre otras.

Pero si se tiene que destacar alguna, es obviamente Visual Studio Code. Este entorno de desarrollo integrado es el más completo en cuanto a desarrollar de forma cómoda gracias a su increíble interfaz.

Se ha elegido este entorno por muchos motivos que se irán abordando poco a poco. Se podría haber escogido tanto Eclipse como Netbeans, pero son más tediosos, no tan modernos y sobre todo no son tan flexibles como Visual Studio Code.

Este fue el entorno de desarrollo integrado que más se usó durante la universidad, principalmente porque era el que se tenia que usar en las asignaturas de programación en Java. Por lo tanto, se tenía más experiencia previa con él y se conocía bien su funcionamiento. Sin embargo, también era tiene sus limitaciones y carencias, especialmente cuando se trata de trabajar con tecnologías más modernas o diferentes del ecosistema Java.

Una de las primeras cosas que se notan al usar NetBeans es que consume bastantes recursos del sistema. Desde el mismo momento en que lo abres, ya se percibe que es un entorno algo pesado, que tarda bastante tiempo en cargar completamente. Esto se debe, en gran parte, al uso elevado de memoria RAM que necesita para funcionar con fluidez. Si se tiene un ordenador con unas especificaciones bajas, esto puede convertirse en un problema serio, afectando al rendimiento general del sistema mientras se trabaja en un proyecto.

También otra cosa para tener en cuenta son los plugins limitados que ofrece. Al no tener tanta comunidad activa su catalogo de plugins y extensiones es bastante limitado, haciendo que sea una muy mala opción para determinados proyectos y sobre todo en este que se usan tecnologías bastantes específicas como Modsecurity.

Tiene poca flexibilidad para otros lenguajes. Es verdad que, si acepta php, pero su rendimiento no llega a ser óptimo y ya si le juntas con un poco de JavaScript no llega a ser un entorno con suficiente rendimiento. En la universidad si se usaba con Java, su rendimiento era mayor, pero debido a que es nativo de este lenguaje.

Otro factor desmotivador, es su diseño, para el gusto de muchos, algo anticuado. Visualmente la hace poco atractiva que otros editores mas modernos. Si se combina esa interfaz antigua con sus actualizaciones lentas, al final hace una experiencia muy pesada y aunque este en manos de Apache no fue para nada buena elección para tener en cuenta.

Aunque, también hay que destacar sus cosas buenas. Puede integrar bases de datos como MySQL y es sencillo de usar si se usa un lenguaje nativo. Pero a la hora de combinarlo con Docker requiere más configuración y es mas tedioso que con Visual, ya que incluye un terminal integrado.

Debido a todo esto, se decidió no elegir Netbeans como entorno integrado para el proyecto y optar mejor por Visual Studio Code, que daba más flexibilidad y una mejor experiencia de desarrollo moderna.

Eclipse fue otro entorno alternativo que se tuvo en cuenta a la hora de usarlo para el proyecto. Durante la universidad fue el primero que se usa en las primeras asignaturas que das en la carrera. Es un IDE bastante conocido y usado en entornos profesionales, sobre todo en empresas que trabajan con java. Y a pesar de haber experiencia con este entorno, no fue la mejor opción para hacer el TFG.

Para empezar, es un entorno bastante pesado y complejo. Tiene muchas funcionalidades, pero están mas pensadas a desarrollos muy específicos o mas corporativos. Haciendo que proyectos mas sencillos o enfocados a tecnologías como php, HTML o JavaScript resulta algo excesivo.

Al igual que Netbeans, también tiene un consumo elevado de memoria RAM y el tiempo de abrir la aplicación o compilar ciertos proyectos en este IDE (Aun siendo java) sea muy tedioso si el ordenador no es muy potente.

Hay que tener en cuenta que Eclipse no esta enfocado a un desarrollo web moderno. Puede configurarse para trabajar con HTML, CSS, JS y PHP, pero hacerlo requiere instalar varios plugins, configurar ciertas rutas y en general preparar ciertas configuraciones que en otros entornos no es necesario o se hacen en un click.

Por ejemplo, en este proyecto, incluye el uso de Docker, base de datos relacional y un WAF (Modsecurity), configurar este entorno para gestionar todo esto se vuelve innecesariamente complicado.

Su interfaz es parecida a la de Netbeans y poco amigable. Es funcional y permite su personalización, sin embargo, puede llegar a ser confusa si es la primera vez. A diferencia de otros entornos, Eclipse tiene una curva de aprendizaje mas alta y esto quita tiempo de enfocarte en el proyecto como tal.

Pero hay que destacar que no todo es negativo, tiene bastante potencia en proyectos grandes con Java, ya que tiene muchas librerías y su sistema de compilación es muy completo. También permite implementar git, framework e incluso conexión a base de datos.

En resumen, Eclipse no fue la mejor opción para el proyecto, ya que no se adapta bien a las tecnologías que se quieren usar. Está más enfocado en proyectos desarrollados en java, sino su configuración se vuelve más tediosa que con Visual Studio.

Visual Studio Code ha sido el entorno de desarrollo escogido para desarrollar el TFG. Es un IDE desarrollado por Microsoft, gratuito y multiplataforma. Desde antes de empezar el TFG se sabia que era la mejor opción, tanto por experiencia personal como por lo bien que combina lenguajes y tecnologías.

La principal razón fue lo rápido y ligero que es. A diferencia de otros entornos, no necesita ninguna instalación pesada ni consume grandes recursos. Si se trabaja en portátil, permite seguir un trabajo ágil.

La personalización que ofrece mediante la instalación, de las llamadas extensiones, es muy completa. En este proyecto, se necesitaron estas extensiones para trabajar con PHP, HTML, CSS y CSV (Excel).

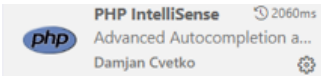

Figura 3.4: Extensión PHP

Figura 3.5: Extensión HTML y CSS

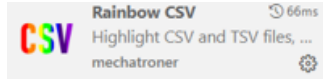

Figura 3.6: Extensión CSV

También existen otras muy útiles como la de Docker (poder ver dentro de los contenedores): poder gestionar sus archivos internos y tener un control total de que ocurre dentro de cada contenedor. Es una extensión muy importante, sobre todo cuando en el Dockerfile se copian archivos del proyecto a dentro del contenedor.

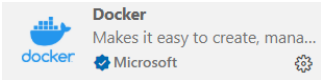

Figura 3.7: Extensión Docker

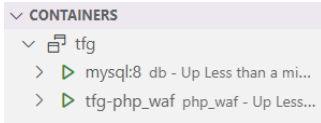

Figura 3.8: Extensión contenedores

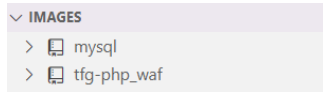

Figura 3.9: Extensión imágenes

Además de la extensión de Docker, existe otra también muy útil, llamada 'Remote Development', puede confundirse con la de Docker, pero tiene una diferencia. Da la opción de abrir el entorno de Visual dentro del propio contenedor. Si quieres ver de cerca, lo que ocurre dentro, esta extensión es ideal.

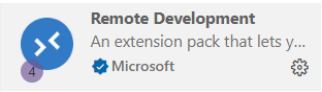

Figura 3.10: Extensión Retome Development

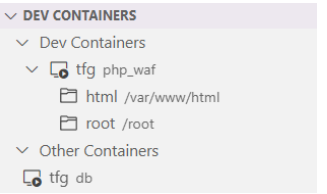

Figura 3.11: Extensión dev containers

Lo que más llama la atención de este entorno integrado es la facilidad de tener todo lo necesario sin salir de él. Puedes ejecutar comandos desde su propia terminal integrada y tener todo lo necesario en un solo entorno. Esta es la mayor ventaja que tiene frente a otros IDE's.

```
PS C:\Users\Myke\Desktop\TFG\TFG> docker compose up -d
[+] Running 2/2
 ✓ Container db       Started
 ✓ Container php_waf  Started
PS C:\Users\Myke\Desktop\TFG\TFG>
```

Figura 3.12: Terminal integrado de Visual Stuido Code

Con este terminal se pueden levantar, construir o bajar los contenedores o hacer pruebas con curl.exe para realizar peticiones a la web sin salir del entorno

Otro punto muy fuerte es su integración con Git. Se Pudo conectar el repositorio de GitLab en unos pocos pasos, para controlar las versiones del proyecto, ya que, se trbajó tanto en un ordenador de sobremesa como desde un portátil, ofreciendo poder trabajar y continuar el trabajo en cualquier momento desde cualquier sitio. Solo se tuvo que instalar git en el ordenador y declarar la ruta en donde guardar el repositorio, a partir de ahí Visual hace el resto.

En cuanto a la estructura del proyecto Visual la facilitó gracias a su explorador de carpetas que permite abrir varios archivos a la vez sin importar su extensión. En este caso se tenían separados los archivos de configuración del WAF, los estilos CSS, los archivos PHP y la configuración de PayPal. Esto es clave para trabajar en un entorno ordenado

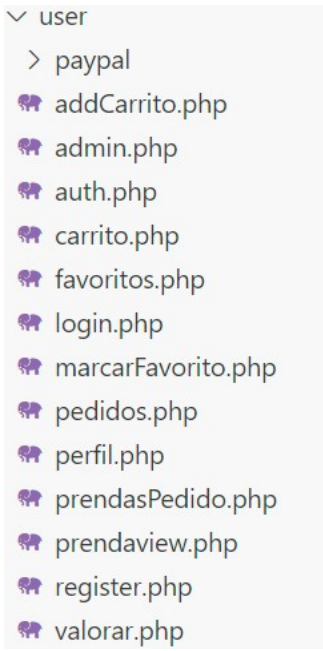

Figura 3.13: Carpeta user

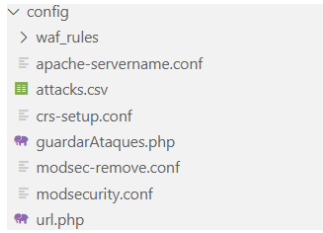

Figura 3.14: Carpeta config

Figura 3.15: Carpeta public y db

En resumen, Visual Studio Code ha sido sin ninguna duda la mejor elección para este proyecto. Permitió integrar todas las tecnologías necesarias de una forma rápida y sencilla, siendo lo que se buscaba desde un primer momento para este TFG. A nivel de productividad fue el que mejor se adaptó a lo que se necesitaba y hoy en día es la opción mas recomendable para usar tanto personal como corporativamente.

3.4. Base de datos

Para el desarrollo de la base de datos de este proyecto se optó por usar como gestor MySQL, principalmente porque es una tecnología muy reconocida y ya se ha trabajado antes con ella tanto en la universidad como en las practicas de empresa.

Desde un primer momento se tenia claro que la estructura de la base de datos iba a ser relacional y entre todas las opciones MySQL era la opción perfecta. Ya que MySQL es un gestor de base de datos muy consolidado con un esquema de tablas relacional con claves primarias y foráneas. A parte de que era el modelo que encajaba perfectamente en la lógica de la web. Las tablas facilitaban muchísimo la estructura de los datos y con PHP se accedía muy rápidamente a la información.

Una se las razones por las que se ha usado este gestor son debido a la experiencia aprendida en la carrera y prácticas, junto a las consultas aprendidas en desarrollo y gestión de bases de datos. Es verdad que en esa asignatura se daba PostgreSQL y en sistemas y servicios web se daba MariaDB, pero no se tenia el conocimiento suficiente que con MySQL. Son muy potentes, robustas y podían perfectamente satisfacer las necesidades de mantener los datos de la web, pero MySQL tuvo un papel más fundamental en el ultimo año de carrera, entendiendo mejor sus conceptos y permisos que las otras dos opciones.

Otra gran ventaja por la que se escogió MySQL fue lo fácil de usar y compatible que era frente a las tecnologías que se querían usar. Sobre todo, en Docker, ya que tiene muy buena integración con los contenedores y tiene imágenes muy actualizadas. Pude hacer un contenedor a parte con la base de datos y se comunicaba sin problema con el otro contenedor, como bien se explicó en el apartado de Docker.

Como se puede ver en el Dockerfile, con una simple línea (`RUN docker-php-ext-install mysqli pdo_mysql`), se puede instalar dos extensiones necesarias para que PHP se comunique sin problemas con la base de datos.

- Mysqli: Permite conectarte a la base de datos con orientación a objetos.
- Pdo_mysql: Mas flexible si el proyecto cambia de base de datos en el futuro.

En definitiva, MySQL daba más seguridad que otros gestores. Tanto al integrarlo en otras tecnologías como la experiencia que se tenia previamente. Permite trabajar cómodamente sin preocupaciones ni problemas de compatibilidad entre tecnologías diferentes.

3.5. Plataforma de pagos

En el desarrollo de la web, era muy importante implementar una pasarela de pagos que diera cierta confianza cuando un usuario realice una compra. Ya que en la web desarrollada, se iba a incluir venta de artículos. Es vital que los usuarios estén cómodos y tenga una buena experiencia al realizar una compra.

Por un lado, se pensó hacer una plataforma sencilla de pagos, pero a la hora del cronograma, se iba un poco del tiempo establecido. Asique se optó por investigar la integración de una API ya hecha.

La primera opción que se probó fue el entorno de desarrollador que ofrece PayPal (Sandbox), en donde permite crear cuentas, gestionar el saldo y en general da una visión profesional de cómo se vería PayPal en tu proyecto.

Account name	Apps	Type	Country	Date created	Settings
MiguelMiguel@business.example.com Default	1	Business	ES	06/05/25, 18:25	⋮
julianjulian@personal.example.com Default	0	Personal	ES	06/05/25, 18:25	⋮

Figura 3.16: Cuentas PayPal

Pero también se investigaron otras plataformas como Stripe, Redsys o Square:

- Stripe: Muy moderna y potente. Tenía mucha personalización y es muy usada en entornos profesionales. Pero requiere más configuración e integrarlo con PHP no era tan fácil

- Redsys: Bastante usada en España por bancos. Pero es bastante más complejo y requiere validaciones al hacer pruebas lo que no es óptimo para un TFG.

- Square: Alternativa moderna y con buen soporte. Integrarlo con PHP es posible, pero esta más orientado a Python, así que se descartó rápidamente.

Finalmente se optó por PayPal, ya que era mucho más conocida por todo el mundo. Se integró perfectamente con PHP y nada más se tenia que instalar un módulo llamado 'composer' desde el terminal (`composer require paypal/paypal-server-sdk:1.0`). Se instala la carpeta vendor para que se pueda acceder a ella, en este caso, desde PHP.

Solo se crearon los dos archivos PHP y otro archivo '.env' donde se almacena toda la lógica de redirección a PayPal y sus tokens.

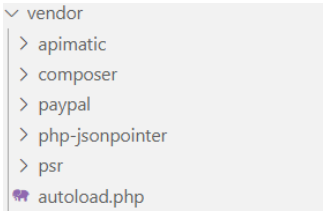

Figura 3.17: Carpeta Vendor

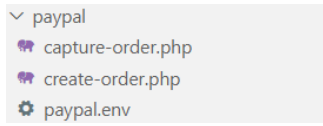

Figura 3.18: Carpeta PayPal

Figura 3.19: Composer

En conclusión, el sandBox de PayPal fue la opción más rápida y funcional para el contexto de este proyecto. Permitió cumplir con el objetivo de integrar una pasarela de pagos sin intervenir negativamente al desarrollo de la web y a la configuración del WAF.

Capítulo 4

Análisis

4.1. Requisitos funcionales

Son las especificaciones que describen las funcionalidades y comportamientos que un sistema debe proporcionar. Es decir, lo que el sistema debe hacer para cumplir con las necesidades del usuario.

1. El sistema deberá permitir a los usuarios el registro con email y contraseña.
2. El sistema deberá permitir a los usuarios iniciar y cerrar sesión.
3. El sistema deberá permitir a los usuarios actualizar su información personal.
4. El sistema deberá permitir a los usuarios explorar el catálogo de ropa disponible, mostrando detalles, imágenes, estado, precio...
5. El sistema deberá permitir a los usuarios añadir prendas a favoritos.
6. El sistema deberá permitir a los usuarios eliminar prendas de favoritos.
7. El sistema deberá permitir a los usuarios añadir prendas al carrito de compra.
8. El sistema deberá permitir a los usuarios eliminar prendas del carrito de compra.
9. El sistema deberá permitir a los usuarios consultar los pedidos realizados.
10. El sistema deberá integrar un sistema de pago que permita realizar compras de forma segura y rápida.
11. El sistema deberá permitir a los usuarios a valorar los pedidos realizados tras su compra.
12. El sistema deberá permitir a los administradores poder consultar los usuarios registrados en el sistema.

13. El sistema deberá permitir a los administradores poder consultar las prendas registradas en el sistema.

14. El sistema deberá permitir a los administradores poder consultar las valoraciones realizadas por los usuarios.

15. El sistema deberá permitir a los administradores poder consultar los ataques registrados en el sistema.

4.2. Requisitos no funcionales

Son las especificaciones que describen las propiedades y características del sistema, tales como rendimiento, usabilidad, seguridad y rendimiento. No definen comportamientos específicos, sino como debe comportarse el sistema.

1. El sistema deberá guardar las contraseñas de los usuarios en forma cifrada (hashed) cuando se registre.

2. Al menos el 80 % de los usuarios evaluarán la interfaz como fácil de usar e intuitiva.

3. El tiempo de respuesta en las transacciones con el sistema de pagos externo (sandBox de PayPal) debe ser menor a 5 segundos en el 95 % de las ocasiones.

4. La aplicación web deberá ser compatible con los principales sistemas operativos móviles (Android e IOS) y navegadores (Mozilla, Chrome, Safari...)

5. El sistema usará MySQL8 como gestor de bases de datos.

6. La aplicación web deberá estar accesible el 99 % del tiempo, exceptuando el tiempo de mantenimiento.

4.3. Reglas de negocio

Son las políticas, cálculos y restricciones que limitan el funcionamiento del negocio de la web. Estas reglas condicionan el comportamiento del sistema para asegurar que se cumplan las normas descritas.

1. El sistema deberá calcular el precio del carrito de compra como la suma del precio de todas las prendas dentro de él.

2. El sistema no permitirá registrar múltiples cuentas con el mismo correo electrónico.

3. Cada producto estará disponible solo una única vez. Una vez comprado, dicho producto desaparecerá.

4. Cuando un usuario cierre sesión, todos los productos que tuviera en el carrito de compra, volverán a estar disponibles para otros usuarios.

5. Todo intento de acceso malicioso detectado por el WAF el sistema deberá almacenarlo.

4.4. Casos de uso

Describen las funcionalidades e interacciones que tiene un usuario o actor con el sistema, alcanzando los objetivos propuestos.

Figura 4.1: Diagrama de casos de uso

Cada caso de uso tiene una secuencia normal de acciones que el sistema realiza según lo introducido por el usuario, detallando el comportamiento que se espera en cada uno.

Como se puede observar un admin (que hereda de usuario) puede ver cierta información importante sobre la web, como administrar todos los usuarios, prendas y valoraciones. Pero también puede ver que intentos de ataques ha habido hacia la web.

Los usuarios, cualquier persona que entra a la web, puede hacer un uso normal de ella con todas sus funcionalidades básicas que ofrece:

Caso de Uso: Identificarse	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa el botón "Identificarse".
3	El sistema solicita correo y contraseña.
4	El usuario introduce los datos y confirma.
5	El sistema valida los datos.
6	El sistema permite el acceso y redirige a el usuario a la página principal.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página Principal.
4b	El usuario pulsa el botón "Registrarse". Se ejecuta el caso de uso "Registrarse".
5a	El usuario deja en blanco los campos. El sistema informa del error y saltamos al paso 3.
5b	El usuario introduce los datos incorrectos. El sistema informa del error y saltamos al paso 3.

Tabla 4.1: Caso de uso: Identificarse (flujo principal y alternativos)

Caso de Uso: Registrarse	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página "Registrarse".
2	El sistema solicita nombre, correo y repetir la contraseña.
3	El usuario introduce los datos y confirma.
4	El sistema valida los datos.
5	El sistema permite el acceso y redirige al usuario, al caso de uso "Identificarse".
Flujos alternativos	
1a, 3a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página Principal.
4a	El usuario deja en blanco los campos. El sistema informa del error y saltamos al paso 2.
4b	El usuario introduce un email ya en uso. El sistema informa del error y saltamos al paso 2.
4c	El usuario introduce contraseñas no coincidentes. El sistema informa del error y saltamos al paso 2.

Tabla 4.2: Caso de uso: Registrarse (flujo principal y alternativos)

Caso de Uso: Añadir prendas al carrito	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El sistema muestra las prendas en estado disponible.
3	El usuario escoge una.
4	El sistema muestra información de la prenda.
5	El usuario pulsa “Añadir al carrito”.
6	El sistema cambia el estado de la prenda.
7	El sistema añade esa prenda al carrito de el usuario.
8	El sistema crea un pedido como no realizado.
Flujos alternativos	
5a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página Principal.
5b	El usuario ya tiene la prenda en el carrito. El sistema cambia el estado de la prenda a disponible, el sistema quita la prenda del carrito, el sistema elimina el pedido no realizado y saltamos al paso 4.

Tabla 4.3: Caso de uso: Añadir prendas al carrito (flujo principal y alternativos)

Caso de Uso: Actualizar datos personales	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa el botón “Perfil”.
3	El sistema muestra nombre, correo y opción para cambiar contraseña.
4	El usuario cambia los datos y confirma.
5	El sistema valida los datos.
6	El sistema actualiza los datos.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página Principal.
4b	El usuario pulsa el botón “Cerrar sesión”. Salida: Página Principal.
5a	El usuario deja en blanco los campos. El sistema informa del error y saltamos al paso 3.
5b	El usuario introduce un email ya en uso. El sistema informa del error y saltamos al paso 3.
5c	El usuario introduce contraseñas no coincidentes. El sistema informa del error y saltamos al paso 3.

Tabla 4.4: Caso de uso: Actualizar datos personales (flujo principal y alternativos)

Caso de Uso: Consultar datos de una prenda	
Paso	**Descripción**
1	El usuario accede a la página principal.
2	El sistema muestra las prendas disponibles.
3	El usuario escoge una.
4	El sistema muestra la información de esa prenda.

Tabla 4.5: Flujo principal del caso de uso: Consultar datos de una prenda

Caso de Uso: Añadir Favoritos	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El sistema muestra las prendas en estado disponible.
3	El usuario escoge una.
4	El sistema muestra información de la prenda.
5	El usuario pulsa “Favoritos”.
6	El sistema añade esa prenda a la sección de favoritos de el usuario.
Flujos alternativos	
1a, 3a, 5a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página Principal.
5b	El usuario ya tiene la prenda en favoritos. El sistema quita la prenda de favoritos y saltamos al paso 4.

Tabla 4.6: Caso de uso: Añadir Favoritos (flujo principal y alternativos)

Caso de Uso: Consultar pedidos	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa sobre el botón “Pedidos”.
3	El sistema muestra los pedidos realizados por el usuario.
4	El usuario pulsa en el botón “Prendas” del pedido.
5	El sistema muestra las prendas de ese pedido.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso queda sin efecto. Salida: Página principal.
4b	El usuario pulsa el botón “Valorar”. Se ejecuta el caso de uso “Valorar pedido”.

Tabla 4.7: Caso de uso: Consultar pedidos (flujo principal y alternativos)

Caso de Uso: Realizar Pedido	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa sobre el botón “Carrito”.
3	El sistema muestra las prendas dentro del carrito de el usuario.
4	El usuario pulsa en el botón “Comprar”.
5	El sistema redirige a el usuario a la pasarela de pagos.
6	El sistema crea un nuevo pedido y guarda sus datos.
7	El sistema muestra los pedido realizados.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso queda sin efecto. Salida: Página principal.

Tabla 4.8: Caso de uso: Realizar Pedido (flujo principal y alternativos)

Caso de Uso: Valorar Pedido	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a valorar pedido.
2	El sistema solicita puntuación y reseña.
3	El usuario introduce los datos.
4	El sistema guarda los datos y redirige a el usuario a pedidos.
Flujos alternativos	
1a, 3a	El usuario cancela y el caso de uso queda sin efecto. Salida: Página principal.
3b	El usuario introduce datos en blanco. El sistema informa de error y saltamos al paso 2.

Tabla 4.9: Caso de uso: Valorar Pedido (flujo principal y alternativos)

Secuencia normal de los casos de uso de un administrador:

Caso de Uso: Consultar Usuarios	
Precondición: El usuario debe estar identificado previamente y ser administrador.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa el botón “Administrar”.
3	El sistema muestra las opciones de administrador.
4	El usuario pulsa el botón “Usuarios”.
5	El sistema muestra todos los usuarios de la base de datos.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página principal.

Tabla 4.10: Flujo principal y alternativo del caso de uso: Consultar Usuarios

Caso de Uso: Consultar Prendas	
Precondición: El usuario debe estar identificado previamente y ser administrador.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa el botón “Administrar”.
3	El sistema muestra las opciones de administrador.
4	El usuario pulsa el botón “Prendas”.
5	El sistema muestra todas las prendas de la base de datos.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página principal.

Tabla 4.11: Flujo principal y alternativo del caso de uso: Consultar Prendas

Caso de Uso: Consultar Valoraciones	
Precondición: El usuario debe estar identificado previamente y ser administrador.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa el botón “Administrar”.
3	El sistema muestra las opciones de administrador.
4	El usuario pulsa el botón “Valoraciones”.
5	El sistema muestra todas las valoraciones de la base de datos.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página principal.

Tabla 4.12: Flujo principal y alternativo del caso de uso: Consultar Valoraciones

Caso de Uso: Consultar Ataques	
Precondición: El usuario debe estar identificado previamente y ser administrador.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa el botón “Administrar”.
3	El sistema muestra las opciones de administrador.
4	El usuario pulsa el botón “Ataques”.
5	El sistema muestra todos los ataques de la base de datos.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso finaliza sin efecto. Salida: Página principal.

Tabla 4.13: Flujo principal y alternativo del caso de uso: Consultar Ataques

4.5. Modelo de dominio

Como en todo Análisis a la hora de crear software, es importante tener un modelo del dominio que represente los objetivos y conceptos de la implementación. En este proyecto hay dos:

- Un modelo orientado al funcionamiento de la web.
- Un modelo para el análisis de los ataques que reciba la aplicación.

Ambos modelos no se relacionan entre ellos directamente ya que cubren distintos objetivos conceptuales. Por un lado, el modelo de dominio de la web se centra en la lógica funcional de la aplicación orientada a un usuario final. Por otro, el modelo de dominio de ataques esta enfocado a las pruebas de la seguridad y refleja las amenazas detectadas.

Se decidió mantenerlos separados, ya que unirlos haría un modelo poco útil desde el punto de vista de diseño, ya que cada uno cumple su función dentro del proyecto y tener un análisis bifurcado da un enfoque mas claro y especializado.

Antes de mostrar ningún diagrama, es importante entender que un modelo de dominio es una abstracción del sistema. La idea no es ver como implementar el software, sino de entender las relaciones y conceptos dentro del dominio. Un modelo de dominio es conceptual, es decir, referenciamos objetos (Modelo Objeto). Un objeto modela una parte de la realidad y delimita un estado y comportamiento con operaciones y métodos (Los métodos se verán en el siguiente apartado, Diagrama de secuencia).

Cada objeto tiene una identidad que lo hace único, no solo sirve con el nombre, se comunican con paso de mensajes. Son instancias de una clase que, en consecuencia, lo que modelamos en el dominio.

Por tanto, este modelo no va a representar código ni la base de datos directamente, no es su objetivo, lo que se quiere conseguir es una relación entre análisis y diseño para entender el dominio antes de implementarlo.

El modelo de dominio de la web, representa las clases esenciales que forman parte de la lógica funcional de la aplicación desde la vista de un usuario:

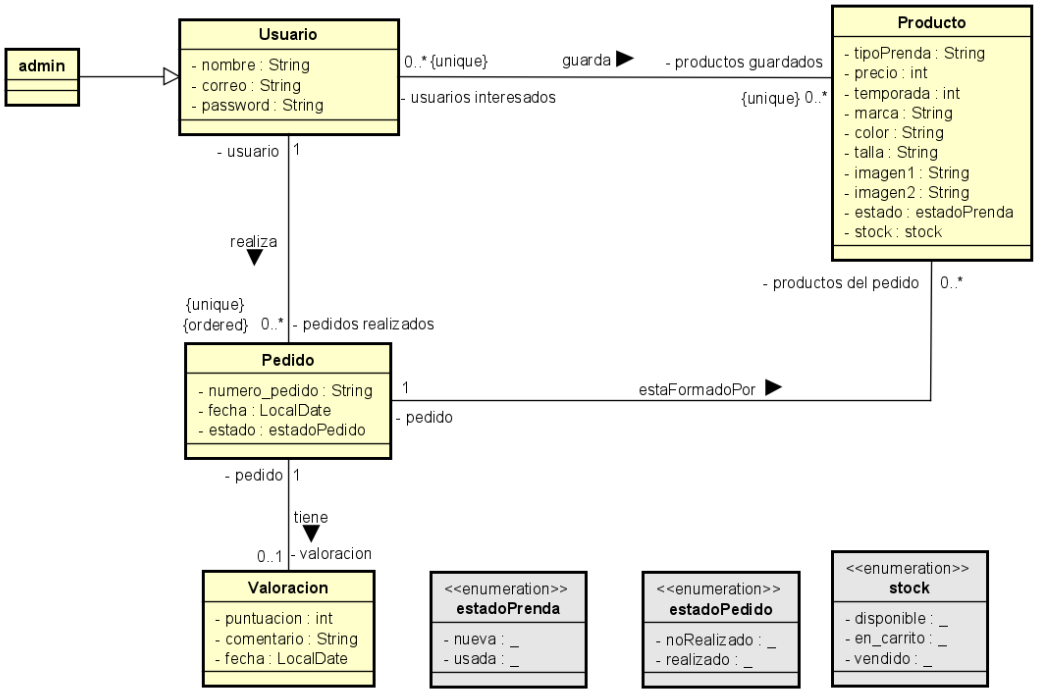

Figura 4.2: Diagrama de dominio (Web)

Usuario: usuario que interactúa con el sistema pudiendo comprar productos, guardarlos y valorarlos. Cada usuario es único.

Admin: Tipo de usuario que interactúa con el sistema con privilegios.

Producto: Productos que se venden en la web, pueden ser guardados por usuarios o comprados. Cada uno de estos tiene estado y stock, este es único (Prendas de reventa).

Pedido: Pedido que realiza un usuario, si no se ha realizado actúa como carrito (Estado pedido). cada pedido es único y esta ordenado por fecha.

Valoración: Valoración que pone un usuario al pedido que ha realizado.

Una vez visto el modelo de dominio de la web, se abordará ahora el modelo de los ataques. Este modelo es mucho mas simple, ya que solo se centra en registrar los ataques recibidos.

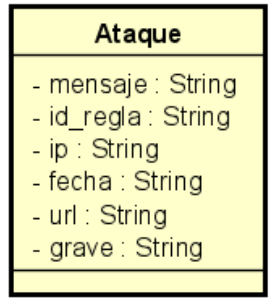

Figura 4.3: Diagrama de dominio (Ataques)

Solo consta de una única clase llamada Ataque, esta es necesaria para tener un control de como se atacará a la web. Registra:

- Mensaje con el tipo de ataque.
- Id de la regla que activó la defensa.
- La IP de donde han hecho el ataque.
- La fecha del momento del ataque.
- La URL introducida con la que se intentó acceder.
- Gravedad del ataque.

Aun que el modelo es pequeño, su función es clave en el objetivo de este proyecto, ya que con este modelo se evalúa la eficiencia de la protección de la web. Y al ser así de simple se entiende mucho mejor como se comporta la aplicación ante diferentes ataques.

4.6. Diagrama de secuencia (Usuario-sistema)

Una vez se tiene el modelo de dominio, se necesita el modelo de interacción, para ver cómo se comunican los objetos entre sí. En este caso se usará el modelo de dominio de la web, ya que este apartado representa la interacción del usuario con el sistema.

Se debe hacer un diagrama de secuencia por cada caso de uso. En esta memoria se desarrollará solo el caso de uso de realizar pedido junto con el de valorar pedido, para representar bien como se comunica el sistema con los usuarios.

Los diagramas de secuencia generan las operaciones de las clases, es por eso que todos los casos de uso deben tener su diagrama de secuencia. Para generar todas las operaciones de las clases.

Pero como bien se ha dicho, solo se seguirán estos dos:

Caso de Uso: Realizar Pedido	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa sobre el botón “Carrito”.
3	El sistema muestra las prendas dentro del carrito de el usuario.
4	El usuario pulsa en el botón “Comprar”.
5	El sistema redirige a el usuario a la pasarela de pagos.
6	El sistema crea un nuevo pedido y guarda sus datos.
6	El sistema muestra los pedidos realizados.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso queda sin efecto. Salida: Página principal.

Tabla 4.14: Caso de uso: Realizar Pedido (flujo principal y alternativos)

Caso de Uso: Valorar Pedido	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a valorar pedido.
2	El sistema solicita puntuación y reseña.
3	El usuario introduce los datos.
4	El sistema guarda los datos y redirige a el usuario a pedidos.
Flujos alternativos	
1a, 3a	El usuario cancela y el caso de uso queda sin efecto. Salida: Página principal.
3b	El usuario introduce datos en blanco. El sistema informa de error y saltamos al paso 2.

Tabla 4.15: Caso de uso: Valorar Pedido (flujo principal y alternativos)

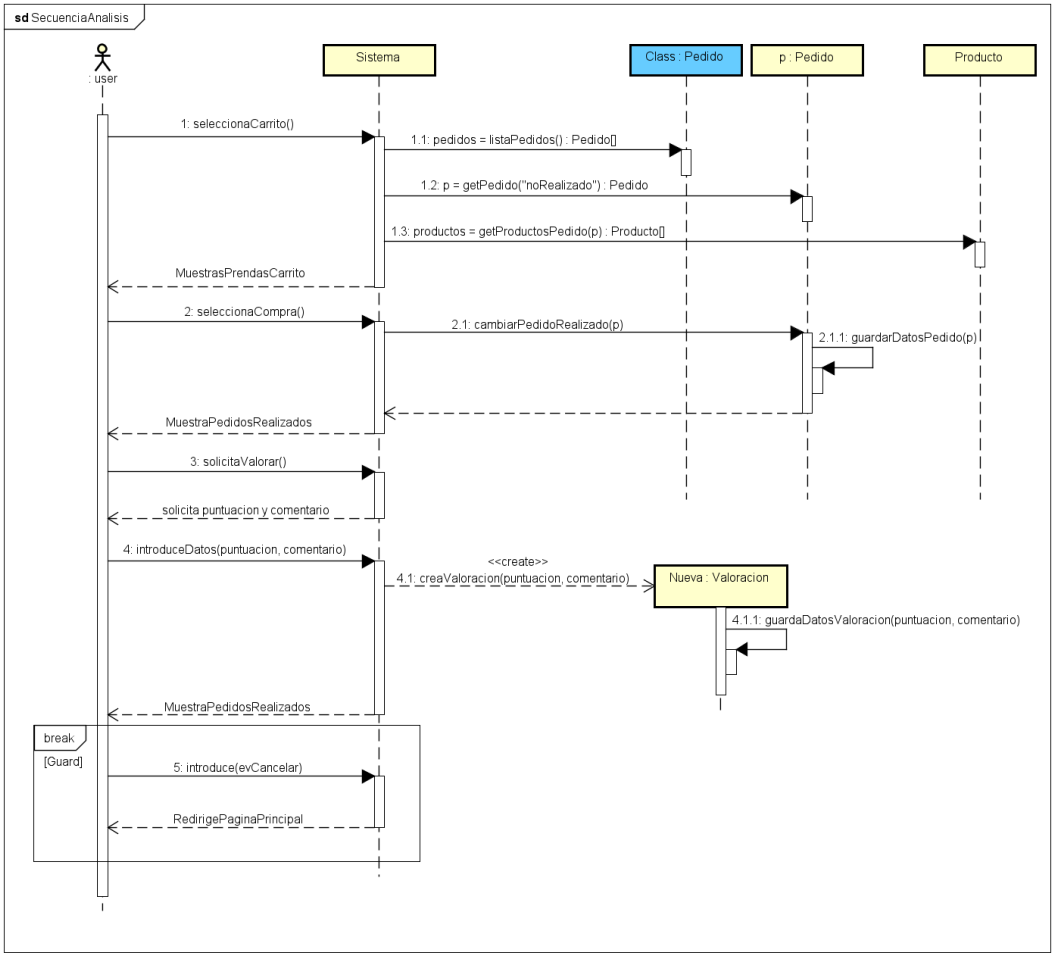

Figura 4.4: Diagrama de secuencia Análisis

Como se puede observar, estos dos casos de uso guardan una fuerte relación y se pueden desarrollar seguidos. En este diagrama se muestra la interacción en función del tiempo de los objetos que participan en los casos de uso. Todos estos pasos de mensajes entre objetos, muestran las operaciones que debe tener cada clase.

Por ultimo, se muestra el modelo de dominio de la web con estas nuevas operaciones en función de las clases implicadas en el diagrama de secuencia.

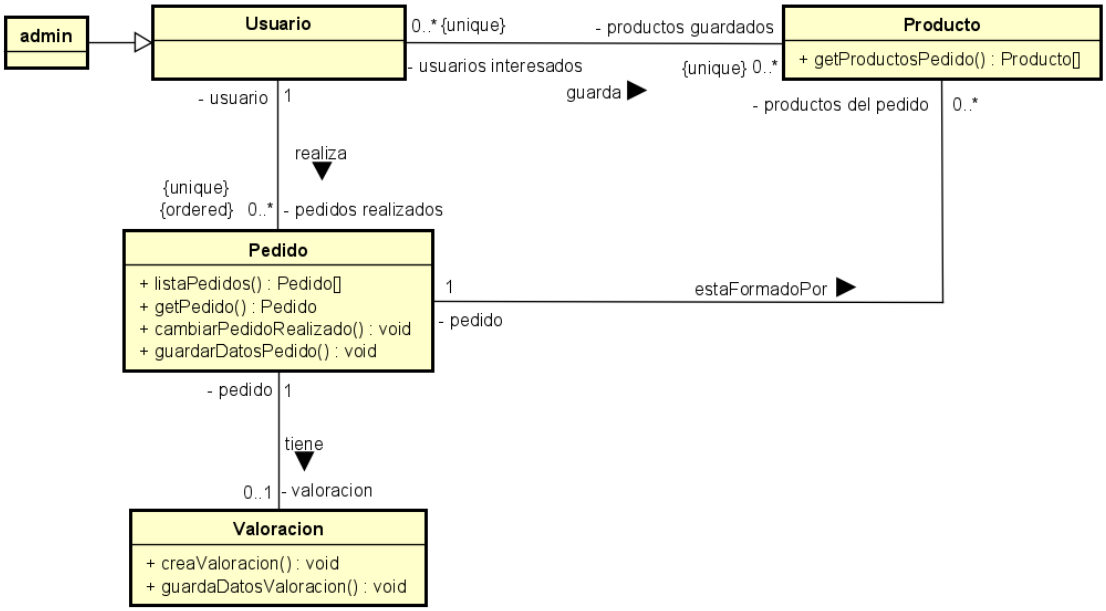

Figura 4.5: Modelo de dominio (Solo operaciones)

Capítulo 5

Diseño

5.1. Arquitectura

La arquitectura seguida para este proyecto ha sido pensada para que se adapte perfectamente a un lenguaje web (lado del servidor) como PHP. Durante el curso, en distintas asignaturas como Diseño de Software, se siguió un modelo de arquitectura Modelo Vista Controlador (MVC).

Este modelo es muy potente, ya que permite que el proyecto sea escalable, fácil de modificar y entender. Sin embargo, al tratarse de una aplicación web, el concepto de "modeloresulta no tan adecuado, ya que se hace uso de variables de sesión y acceso a la base de datos.

De este modo, se optó por desarrollar una arquitectura formada por 3 capas principales: Vista, Controlador y persistencia. Esta estructura se asemeja más a un modelo vista controlador plano o degenerado, ya que el modelo no es tan necesario y se puede omitir.

De esta forma, conseguimos una capa intermedia donde se encuentra toda la lógica y actúa como un controlador (Archivos PHP). Una vista donde el usuario podrá interactuar con el sistema (Archivos HTML y CSS). Y por último, una capa de persistencia donde se accede a la base de datos.

Existen otras dos capas, 'config' y 'vendor', que forman parte de la arquitectura general, pero no pertenecen directamente al patrón Vista - Lógica - Persistencia.

Por ejemplo, la carpeta vendor se utiliza para gestionar dependencias externas, como la pasarela de pagos, mientras que config se emplea para configurar parámetros del sistema, como la detección o registro de ataques.

Arquitectura general del proyecto

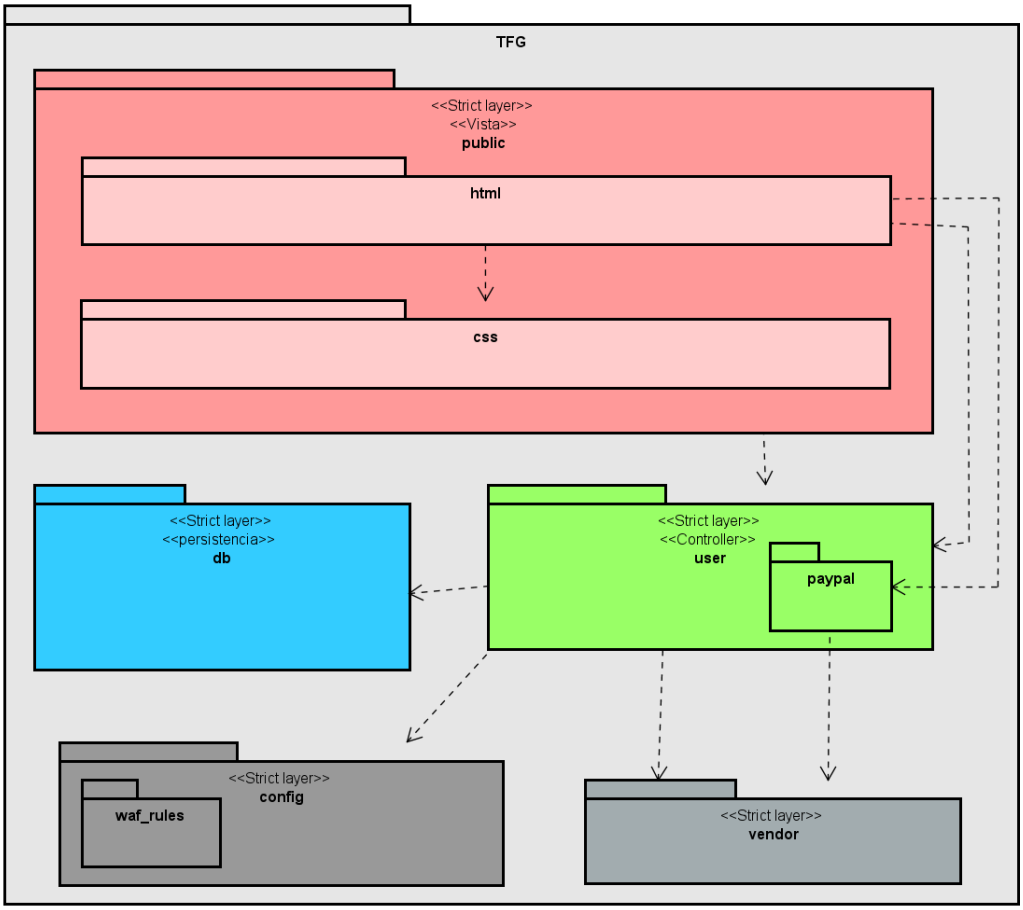

Figura 5.1: Arquitectura General

Todas las capas son estrictas, ya que solo pueden ser accedidas desde una capa superior.

Por cada vista que ve el usuario en la aplicación web, existe un controlador que la proporciona, al igual que la persistencia.

El acceso mediante URL está restringido únicamente al controlador. Con la finalidad de proteger las vistas y la persistencia que puedan ser accedidas directamente por el usuario desde el navegador.

Es el controlador quien recibe la petición del usuario, gestiona la lógica necesaria —incluyendo la interacción con la persistencia si es necesario— y finalmente entrega la vista al usuario.

Esta estructura garantiza una mayor seguridad, ya que impide el acceso directo a recursos sensibles o a vistas que no deberían mostrarse de forma aislada, manteniendo así la separación de responsabilidades y el control sobre el flujo de la aplicación. De todas formas, esta parte se explicará en el capitulo de seguridad.

Ejemplo de tripletes, vista - controlador - persistencia de admin.php:

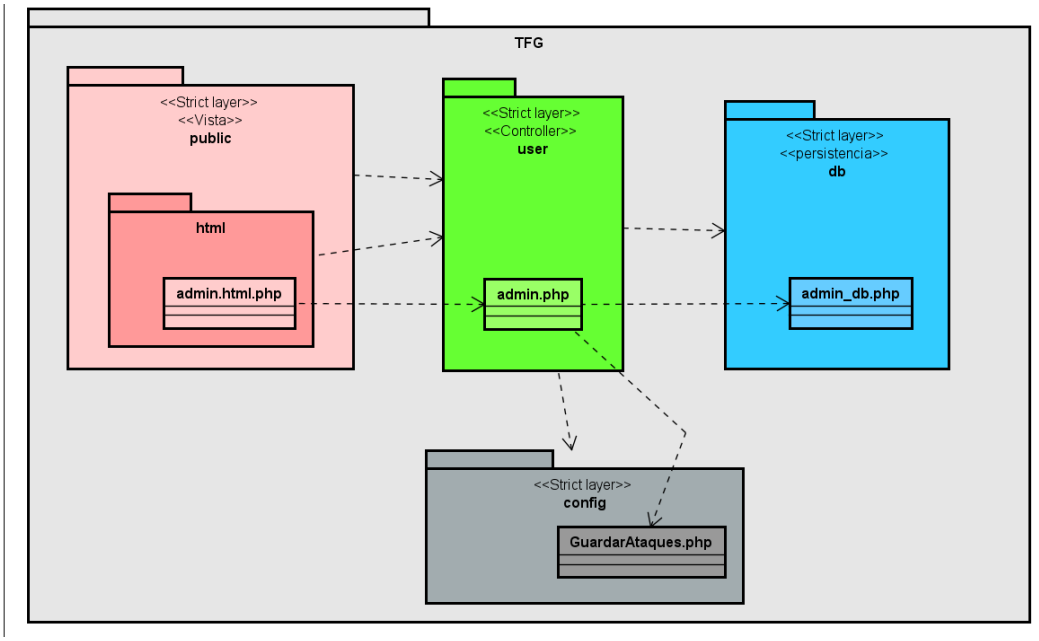

Figura 5.2: Ejemplo arquitectura Admin.php

Una vez explicada la arquitectura del proyecto, es importante incluir un mapa de navegación, ya que está bastante relacionado con la estructura del sistema.

Este mapa permite visualizar cómo se conectan las distintas páginas desde la vista del usuario, mejorando la comprensión del flujo de navegación:

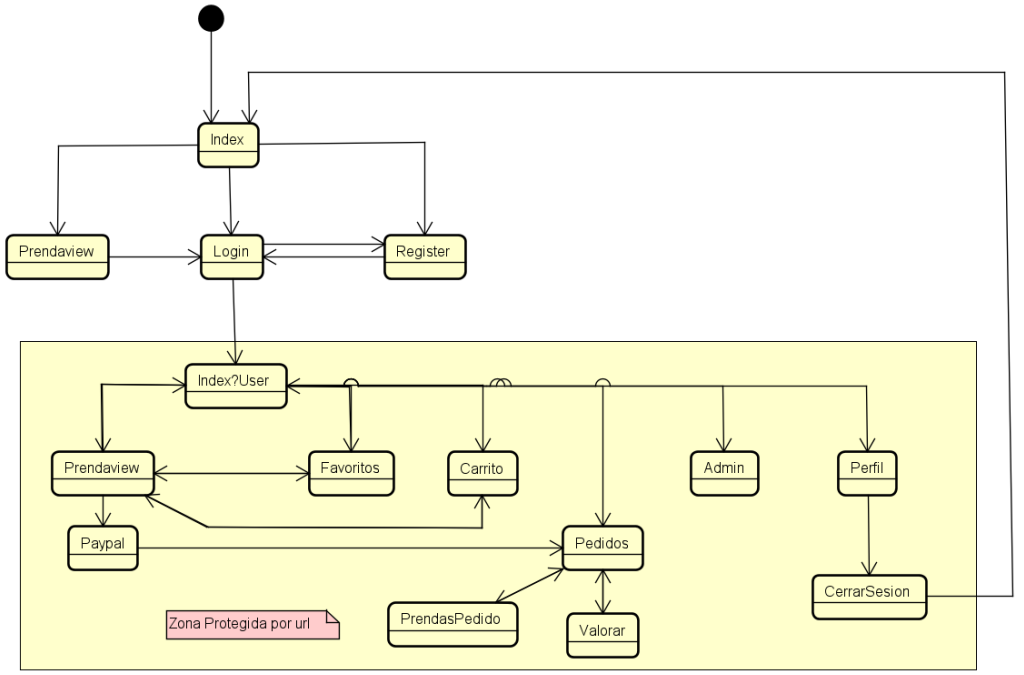

Figura 5.3: Mapa de Navegación

Como se puede observar en él, se modela lo comentado anteriormente. Es importante proteger ciertas rutas para que nadie pueda acceder a ellas mediante url. La zona protegida está delimitada por un rectángulo amarillo y en su interior están las paginas que prohíben entrar sin un registro previo y de forma malintencionada.

Si el usuario sin registrarse intenta entrar, se le redirigirá al login y si mete el id de otro usuario no le dejará obtener sus datos. Esto se abordará más en profundidad en el siguiente capitulo pero es importante destacarlo ya que se tiene el mapa delante.

5.2. Modelo de Datos

En esta sección se abordará, el diseño lógico de la base de datos. Es decir, como se ha pasado del modelo de dominio del análisis a este modelo de diseño. Aquí abordaremos las clases como tablas (EL termino clase y objeto desaparece) y las relaciones como dependencias (ya no hay lineas bidireccionales, ahora hay lineas discontinuas con navegabilidad).

También surjen más atributos como las claves primarias (Atributos de color rojo en el diagrama) y claves foráneas (atributos de color azul en el diagrama). Los atributos de color verde simbolizan que son tanto claves primarias como foráneas.

Otra cosa a destacar, es la implementación de una nueva tabla. Cuando en análisis se tiene una relación muchos a muchos (0..* - 0..*) a la hora de pasar a diseño, en la base de datos se origina una tabla intermedia con el nombre de la relación. En el caso del diagrama entre usuario y producto, surje la tabla favorito.

Los enumerations, deberían estar unidos a la clase que los origina, pero en la forma que se ha diseñado la base de datos no ha hecho falta debido a que no se ha creado una tabla aparte para ellos sino que están dentro de la propia tabla.

```
CREATE TABLE pedidos (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT UNSIGNED NOT NULL,
    numero_pedido VARCHAR(32) NOT NULL UNIQUE,
    fecha DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    estado ENUM('noRealizado','realizado'),
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE
);
```

Figura 5.4: Ejemplo de enumeration

Por último a destacar, se optó por crear una tabla carrito que uniera pedido con producto, haciendo una relación muchos a muchos y generando dicha tabla intermedia. De esta forma se podía modelar que un producto podría estar en varios carritos a la vez. Pero al haber solo una única prenda por producto, era bastante frustrante añadir una prenda al carrito y que otro usuario la compre.

Teniendo en cuenta otras apps de venta (Que reservan el producto cuando se añade al carrito) y sobretodo en la vida real, es raro coger lo del carrito de otro. Se decidió no crear la tabla carrito, dejar la relación 1 a muchos entre pedido y producto y modelar el pedido 'noRealizado' como carrito. (Visto en las reglas de negocio).

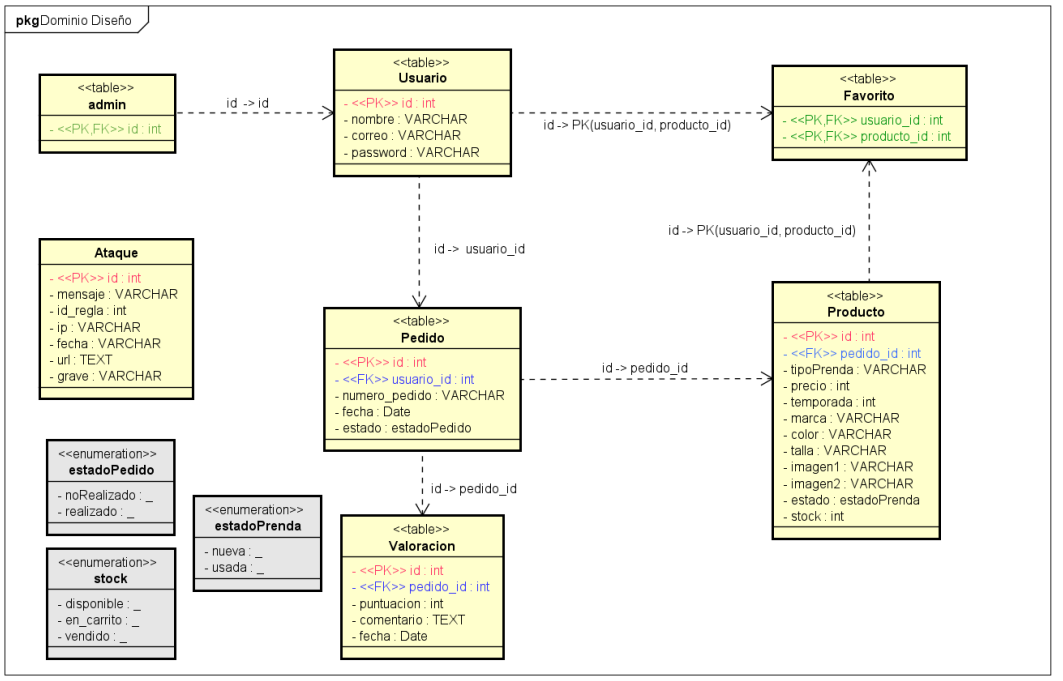

Figura 5.5: Data Model

En análisis se comentó la necesidad de dos dominios separados para cubrir dos necesidades diferentes. En diseño esos dos dominios se diseñan en la misma base de datos. Pero como se ve en la imagen, al final se ha decidido diseñarlos en la misma base de datos. Es verdad que se optó por crear otra base de datos solo con la tabla ataques, pero teniendo en cuenta que era solo una tabla, no tenía sentido hacer otra conexión a otra base de datos, era más cómodo tenerla en el mismo diseño.

Si se hubiera creado otra base de datos aparte, en la clase conexión (Se optó por modelar la base de datos orientada a objetos por escalabilidad y facilidad de uso), se tendría que haber creado otra conexión o tener en cuenta otras configuraciones que para una sola tabla, se vio innecesario.

5.3. Diagrama de secuencia interno

Como en todo diseño de software, es necesario un diagrama de secuencia interno, para ver que alcance tiene la arquitectura. Para ello se usará el mismo caso de uso que se mostró en el diagrama de secuencia, del capitulo de análisis, como ejemplo.

C.U Realizar Pedido.

Caso de Uso: Realizar Pedido	
Precondición: El usuario debe estar identificado previamente.	
Paso	**Descripción**
Flujo principal	
1	El usuario accede a la página principal.
2	El usuario pulsa sobre el botón “Carrito”.
3	El sistema muestra las prendas dentro del carrito de el usuario.
4	El usuario pulsa en el botón “Comprar”.
5	El sistema redirige a el usuario a la pasarela de pagos.
6	El sistema crea un nuevo pedido y guarda sus datos.
6	El sistema muestra los pedidos realizados.
Flujos alternativos	
1a, 2a, 4a	El usuario cancela y el caso de uso queda sin efecto. Salida: Página principal.

Tabla 5.1: Caso de uso: Realizar Pedido (flujo principal y alternativos)

Para seguir un mismo código de colores, las vistas se pintarán de rojo, los controladores de verde y las bases de datos de azul.

El patrón utilizado en el proyecto, en cuanto al acceso a la base de datos, se asemeja un poco al patrón DAO (Data Access Object) en cuanto a separar responsabilidades. Cada página tendrá su vista, controlador y acceso a base de datos. Menos el paquete PayPal que tendrá un mismo archivo de acceso a base de datos (paypal_db.php) para los dos controladores. Se optó hacerlo así para más comodidad.

- Vista: Están marcados en rojo y únicamente se encargan de la presentación e interacción directa con el usuario.
- Controlador: Están marcados en verde y gestionan la lógica de la web, muestran su vista correspondiente al usuario y gestionan la conexión a la base de datos. Hacen de intermediario entre la vista y la base de datos.
- Acceso a base de datos: Están marcados en azul y se realizan en funciones separadas como el patrón DAO, permitiendo consultas desacopladas del controlador.

Utilizando este patrón conseguimos mucha escalabilidad y mantenimiento si en un futuro se quiere añadir código o más implementación.

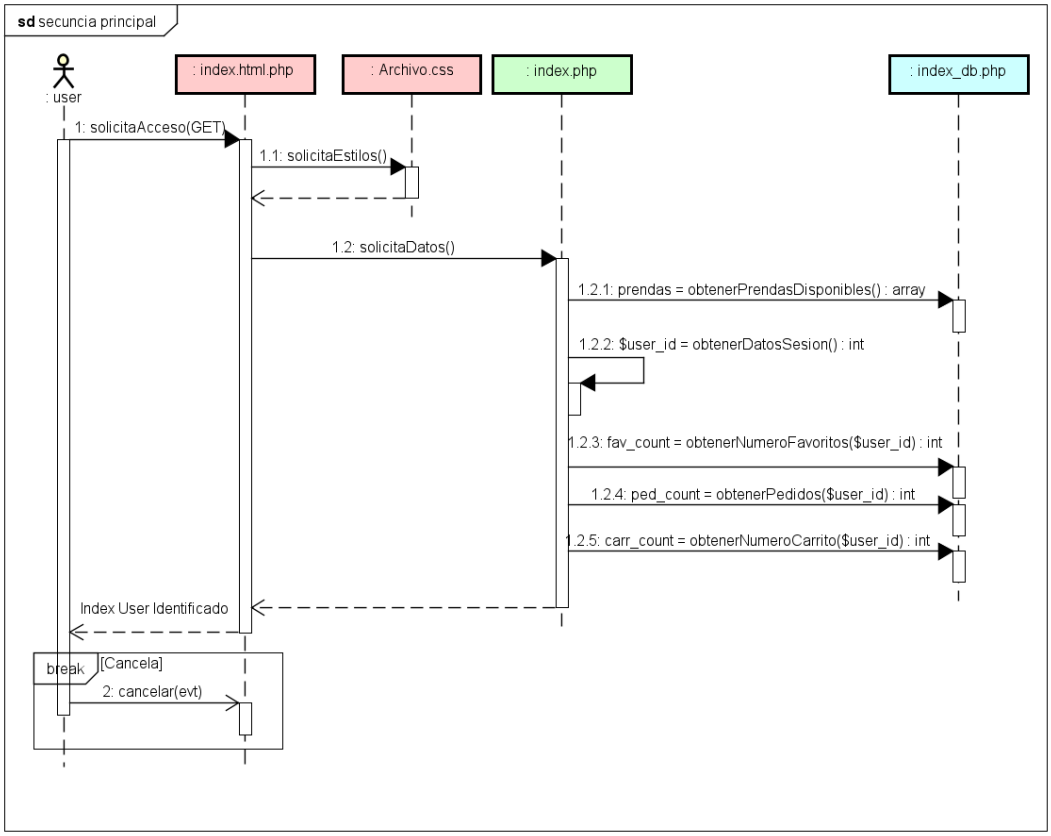

Figura 5.6: Diagrama de secuencia Diseño (Parte 1)

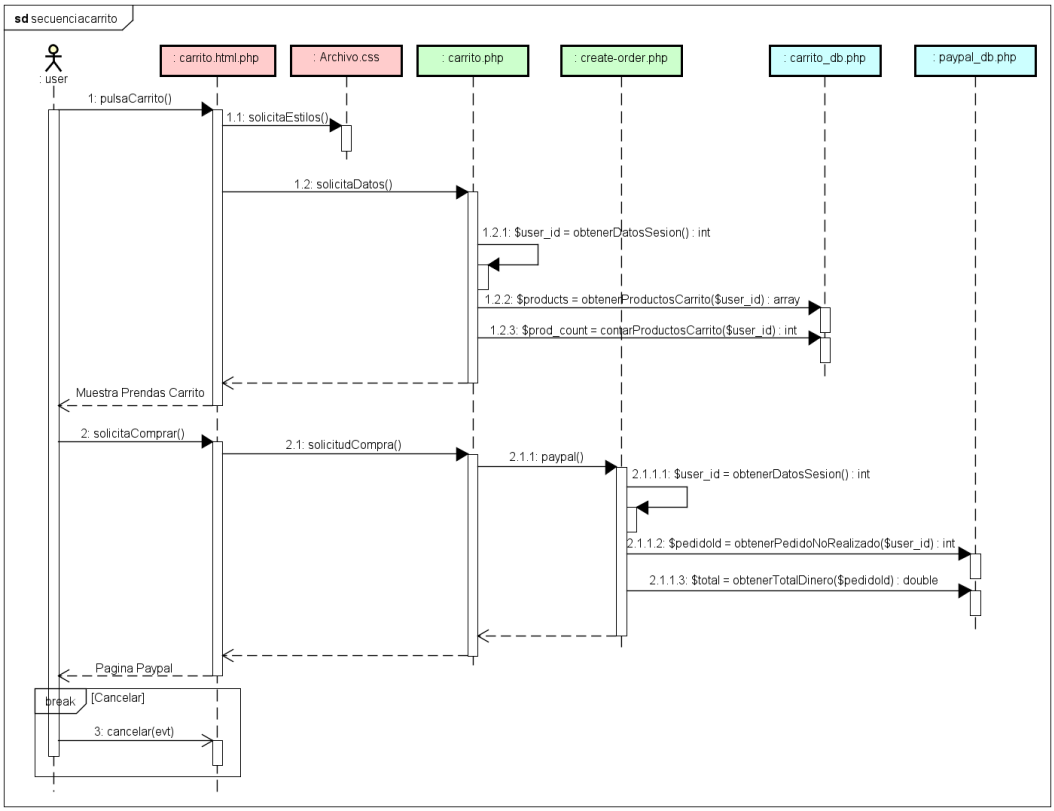

Figura 5.7: Diagrama de secuencia Diseño (Parte 2)

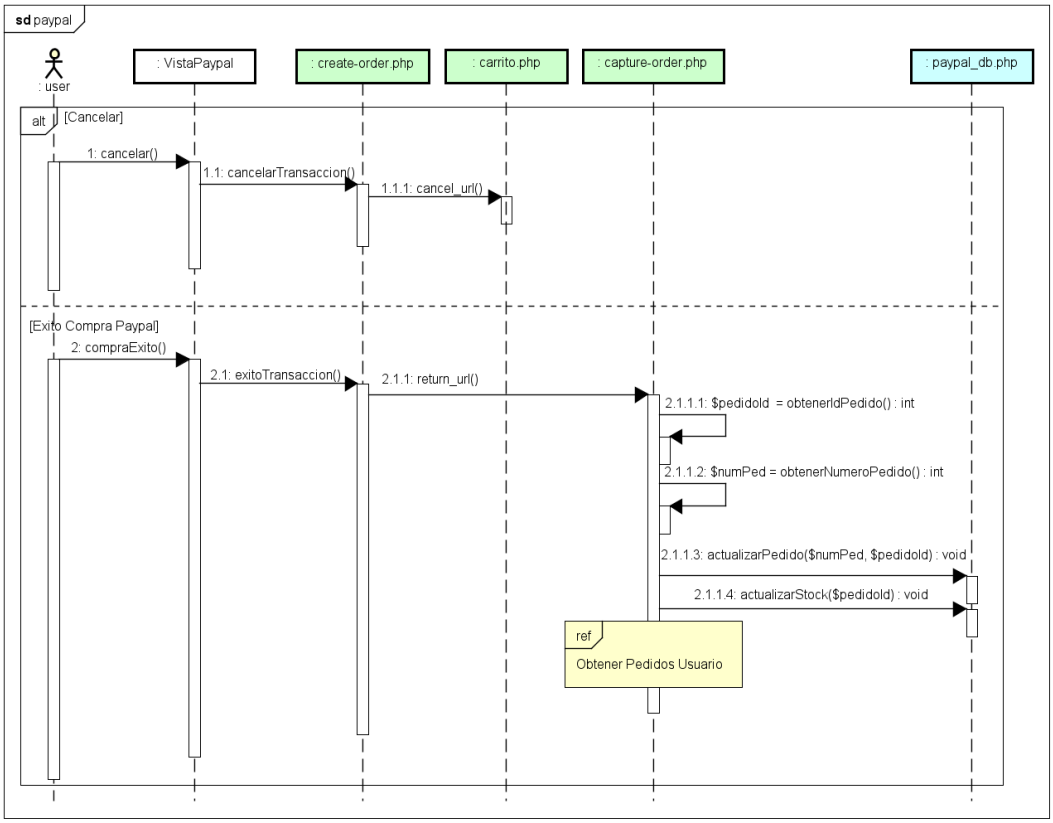

Figura 5.8: Diagrama de secuencia Diseño (Parte 3)

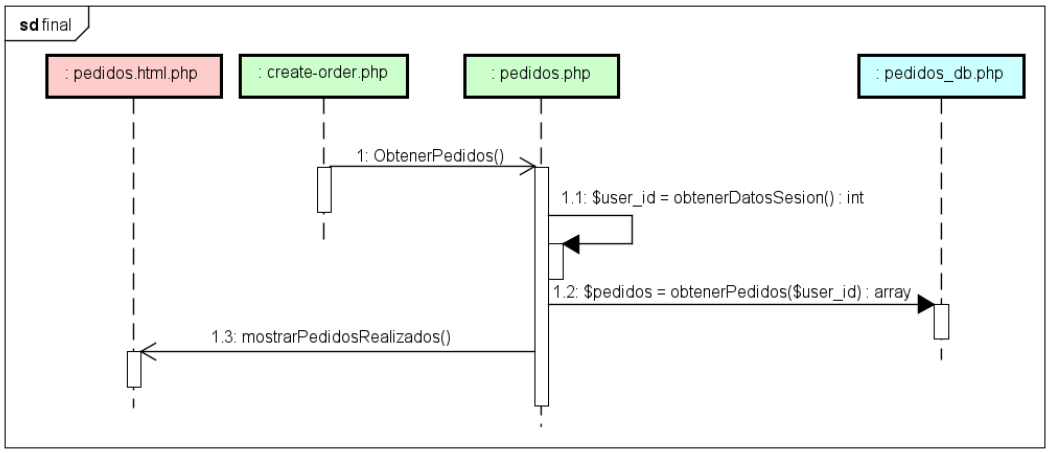

Figura 5.9: Diagrama de secuencia Diseño (Parte 4)

5.4. Diagrama de despliegue

Como último paso en el capitulo de diseño. Se mostrará el diagrama de despliegue. Este es necesario ya que da una visión global sobre que alcance tiene el proyecto en cuando a tecnologías usadas y comunicación entre ellas.

Que 'execution eviroments' se usan y en que contenedores se realizan dichas comunicaciones. Como se puede ver en la imagen siguiente, se puede acceder a la aplicación web desde otro dispositivo (Mediante un túnel que se explicara en el siguiente capitulo). Hay dos 'execution enviroment' y dos 'docker container':

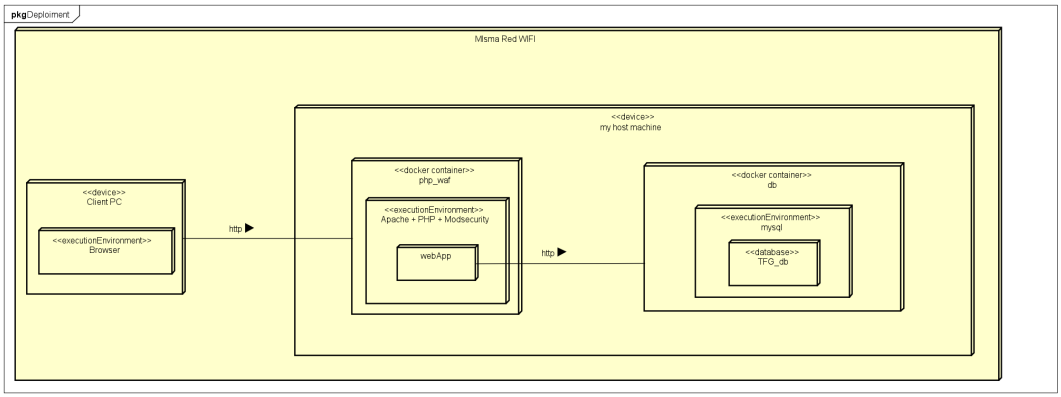

Figura 5.10: Diagrama de despliegue

Las comunicaciones entre el navegador del usuario y el contenedor principal (php_waf). Se realizan mediante peticiones http al puerto 8080, que es el puerto expuesto de este contenedor.

La conexión http del contenedor php_waf a la base de datos no es desde un navegador, ya que en el docker-compose el contenedor de la base de datos no tiene un puerto expuesto. Esto se ha hecho con intención para que nadie pueda acceder por http a nuestra base de datos. Unicamente puede acceder el contenedor del WAF ya que entre ellos dos, hay una red interna que les permite comunicarse, creando una conexión segura y evita exponer el puerto de la base de datos.

Capítulo 6

Seguridad y comparativa

6.1. Que es un WAF y cuáles existen

En este Capitulo, se verán los diferentes WAF que han sido investigados a lo largo de estos 3 meses y el proceso de elección que se ha seguido para introducir uno de estos en el proyecto.

Antes de explicar cada uno de ellos, es importante entender el concepto de WAF. Un WAF es un sistema de seguridad (firewall) para proteger aplicaciones web. Su funcionamiento es filtrar y bloquear trafico que pasa entre el exterior y la web, en este caso solicitudes http malicioso. A diferencia de un firewall tradicional que protegen una zona de la red especifica, un WAF está mas enfocado a proteger una aplicación, defendiendo vulnerabilidades explotables.

Como bien se comentó antes, existen varios ataques comunes que los WAF son muy potentes en mitigar, tales como: Inyección SQL, Cross-Site Scripting (XSS), File inclusión, Fuerza bruta... Entre muchos más. Es por ello que muchos optan por implementar reglas ya definidas de OWASP CRS para conseguir una protección frente a estos ataques.

Como es lógico además de las reglas predefinidas que proporciona OWASP CRS, cada WAF puede crear las suyas propias según la aplicación que este defendiendo. Pueden operar entre diferentes modos, según el grado de efectividad o la responsabilidad que le queramos dar:

- Modo Pasivo (Monitor Only): Con este modo, el WAF solo registrará los eventos y peticiones sin actuar.
- Modo activo (Prevention): En este modo mas restrictivo, el WAF bloquea directamente el trafico que considera sospechoso.
- Modo Mixto (Detection Only): Este modo es el mas común a la hora de configurar un WAF. Permite monitorear reglas personalizadas y luego las aplica activamente.

Una vez sabido que es y que hace exactamente un WAF. Se va a explicar las posibles opciones que se barajaron para implementar finalmente en el proyecto. En total se investigaron tres opciones: BunkerWeb, Coraza y ModSecurity.

BunkerWeb en un web aplication firewall (WAF) de código abierto que fue diseñado para proteger fácilmente aplicaciones web modernas sin una configuración o experiencia en seguridad extremadamente complejas. Su objetivo es centrarse en automatizar su integración en entornos DevSecOps, siendo la mejor opción en entornos que necesitan protegerse de forma inmediata con un personal reducido.

A diferencia de las otras dos opciones que se van a ver, este WAF apuesta por una experiencia "plug-and-play"donde la mayoría de configuraciones vienen ya predefinidas según el entorno que se quiera desarrollar.

Arquitectura y funcionamiento

BunkerWeb funciona con un reverse proxy, con unos servidores web bastante potentes como Nginx o Apache. Su arquitectura se abstrae gracias a ciertas configuraciones que hacen que sea mas simplificada y permite que los servidores web sean protegidos de forma sencilla sin tener que modificar la aplicación.

Se puede implementar mediante contenedores docker gracias a su propia imagen "bunkerity/bunkerweb"que se puede encontrar en la pagina oficial de docker, "Docker hub". Ahí hay una amplia variedad de imágenes que puedes usar para crear tus contenedores según el proyecto que se desee y una buena documentación de que incluyen y como usarla.

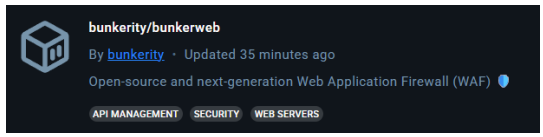

Figura 6.1: Imagen de Búnker Web

Se puede añadir sin necesidad de crear un Dockerfile o directamente desde el docker-compose. Internamente consta de varios módulos que están divididos por tareas las cuales tienen responsabilidades diferentes como filtrar trafico HTTP, aplicar las reglas de OWASP,validar cabeceras... Gracias a este diseño modular, se puede adaptar a cualquier entorno sin necesidad de una configuración muy complicada.

Características principales

Como bien se ha dicho, es ideal para usarlo con docker, ya que dentro de su filosofía, este WAF muestra una seguridad sin fricción. Docker le permite integrarlo rápidamente sin perder tiempo en manuales.

Tiene una interfaz gráfica, que hace mucho mas visual su configuración y para los usuarios que no tienen mucha experiencia en configuración de firewalls en general, es de gran ayuda.

Incorpora las reglas OWASP CRS, son un conjunto de reglas ya definidas como base de protección inicial, reduciendo los falsos positivos y añadiendo contenido como protección contra bots automatizados, ataques DDoS...

También tiene la opción de generar certificados automáticos SSL para que tu entorno se despliegue de forma dinámica, reduciendo la fricción de seguridad HTTPS sin tocar un manual. Aun así es muy recomendable leer los manuales que la web oficial proporciona.

Ventajas

En cuanto a las ventajas, BunkerWeb presenta un amplio margen donde moverte, siendo una de las primeras opciones elegidas por muchos desarrolladores de aplicaciones web. Uno de sus principales puntos fuertes es que elimina la necesidad de escribir reglas complejas manualmente, permitiendo configurar una protección efectiva en cuestión de minutos, incluso con equipos reducidos o con poca experiencia en seguridad.

Es perfecto para entornos y proyectos contenerizados, ya que tiene un diseño modular y una configuración mediante variables de entorno que lo hacen altamente compatible con tecnologías como Docker. Facilitando su despliegue, y mantenimiento en arquitecturas modernas.

Además, BunkerWeb ofrece una interfaz web intuitiva, lo que lo diferencia de otros WAF como Coraza o ModSecurity, que carecen de una GUI integrada. Esta interfaz permite controlar ataques, cambiar configuraciones y ver estadísticas de seguridad de forma mas cómoda y clara.

Otra ventaja a destacar es su capacidad para integrarse fácilmente con otros servicios, muchas veces de forma automática como la gestión de certificados SSL, y su soporte nativo para balancear la carga y otros módulos de seguridad avanzada.

Por último, BunkerWeb se actualiza con frecuencia y cuenta con un buen nivel de automatizar la gestión de amenazas, lo que reduce el trabajo manual y mejora el tiempo de respuesta ante nuevos vectores de ataque.

Limitaciones

Pero también tiene sus desventajas. Al proporcionar muchas reglas ya integradas, la personalización es más limitada en comparación con otros WAF. Si bien permite ajustar ciertas reglas, la flexibilidad a la hora de crear o modificar reglas complejas está restringida, lo cual puede ser un inconveniente en entornos que requieren configuraciones de seguridad muy específicas o adaptadas.

Existe también cierta sensación de çaja negra". Al ofrecer tanta automatización y rapidez en la configuración, se pierde visibilidad sobre el funcionamiento interno del sistema. Esto puede representar una desventaja importante si en un proyecto personal se necesita seguir y auditar detalladamente el flujo de decisiones de seguridad, o analizar con precisión cómo y por qué se bloquean ciertas peticiones.

Además, BunkerWeb, aunque ofrece una interfaz gráfica útil, no permite un acceso tan granular al motor de análisis como otras soluciones más técnicas, como ModSecurity. Esto puede limitar la capacidad de análisis forense en caso de incidentes avanzados.

Otro aspecto a considerar es que, aunque tiene buena integración con entornos contenerizados, no es tan modular a nivel interno como Coraza, y algunas funcionalidades más avanzadas pueden depender de la edición empresarial o requerir configuraciones adicionales.

Finalmente, al ser una solución relativamente más cerrada en cuanto a reglas y funcionamiento interno, su comunidad es más reducida y menos orientada a desarrolladores, por lo que la resolución de problemas complejos puede depender directamente del soporte oficial.

Conclusión

BunkerWeb es muy buena opción debido a su fácil implementación y compatibilidad con tecnologías modernas. Aun que esta hecho para entornos que no requieran reglas muy personalizadas o políticas muy estrictas, si resulta muy útil en proyectos que necesitan una protección eficaz y rápida. Es por eso que este enfoque llamó la atención a la hora de valorarlo como candidato a WAF de la aplicación web desarrollada.

La segunda opción que se tuvo en cuenta fue Coraza. Este WAF de código abierto esta desarrollado en Go y respaldado por la comunidad de OWASP. Fue diseñado como alternativa moderna a la ultima opción que se verá, ModSecurity. Coraza ofrece una arquitectura como la de BunkerWeb, modular. Teniendo un alto rendimiento y compatibilidad con el lenguaje de reglas "SecLandƶ de OWASP.

Su enfoque cuando se desarrolló fue para dar una solución nativa adaptable a entornos modernos como API gateways, sin sacrificar la posibilidad de crear reglas personalizadas.

Arquitectura y funcionamiento

Coraza es una librería embebida, es decir, puede funcionar como un servicio independiente dependiendo de cómo se utilice. Como se explicó anteriormente, su diseño modular permite integrarlo e implementarlo de forma flexible según las necesidades del entorno.

Por ejemplo, puede utilizarse como librería en aplicaciones desarrolladas en Go, integrarse con Caddy (un servidor web alternativo a Apache o Nginx), o funcionar junto a tecnologías compatibles con OWASP. Su integración con estas tecnologías es altamente recomendable, ya que permite ofrecer una protección más robusta frente a vulnerabilidades comunes y ataques web frecuentes.

Características principales

Las características principales y más importantes son la compatibilidad con SecLang y OWASP. Coraza soporta ambos lenguajes de reglas, ofreciendo una protección eficaz contra una amplia gama de ataques.

A diferencia de otros WAF, cuenta con una alta capacidad de personalización, ya que fue diseñado específicamente para ello. Esta personalización no se limita únicamente a las reglas, sino que también permite la creación de plugins para implementar acciones personalizadas, lo que lo hace muy flexible y adaptable a diferentes entornos y necesidades de seguridad.

Ventajas

Gracias a su diseño en Go, este WAF proporciona un alto rendimiento, consumiendo pocos recursos, tanto para aplicaciones con un tráfico elevado como para entornos con recursos limitados.

Una muy buena ventaja que ofrece es un entorno de pruebas llamado Coraza Playground, donde se permite probar y validar las reglas de seguridad implementadas en un entorno controlado. De esta forma, se facilita el desarrollo y puede ser más visual ajustar ciertos parámetros de configuración.

Además, su compatibilidad con el conjunto de reglas OWASP CRS y su soporte para entornos modernos como Kubernetes, Caddy o proxies compatibles con WebAssembly lo convierten en una herramienta muy versátil para arquitecturas actuales.

Limitaciones

Coraza no es tan intuitivo, ya que carece de una interfaz gráfica, y si no se tiene mucha experiencia en el sector, puede llegar a ser algo difícil de entender. Presenta una curva de aprendizaje algo más elevada que otras opciones.

También es una solución relativamente reciente y no hay tanta información acerca de ciertos fallos o de qué hacer si surgen problemas durante su implementación o desarrollo. Es cierto que existe una comunidad, pero de momento no es muy grande, al igual que su documentación, que aún está en crecimiento.

Además, al estar basado en el lenguaje Go y no contar con un panel de gestión visual, puede requerir conocimientos más técnicos para la integración y monitoreo. Esto puede representar una barrera en equipos menos familiarizados con arquitecturas cloud-native o DevOps.

Finalmente, aunque su compatibilidad con OWASP CRS es una ventaja, la configuración avanzada de reglas personalizadas puede resultar compleja sin experiencia previa con el lenguaje SecLang.

Conclusión

Coraza es una solución mas moderna que en unos años será muy elegida debido a todas las ventajas que ofrece. Presenta una evolución considerable en el ámbito de firewalls para aplicaciones web y cubre altamente las necesidades de seguridad actuales. Su compatibilidad con otras tecnologías existentes y su buen rendimiento, lo convierten en una muy buena opción para proyectos que buscan una protección robusta sin romper con eficiencia.

ModSecurity es un firewall de aplicaciones web (WAF) de código abierto, desarrollado inicialmente en 2002 por Ivan Ristić y actualmente mantenido por la comunidad de OWASP. Conocido como la "navaja suiza"de los WAFs, ModSecurity ofrece una amplia gama de funcionalidades para proteger aplicaciones web contra diversas amenazas, permitiendo a los defensores de aplicaciones web obtener visibilidad en el tráfico HTTP(S) y proporcionando un potente lenguaje de reglas y una API para implementar protecciones avanzadas.

Arquitectura y funcionamiento

ModSecurity, a diferencia de los anteriores WAF, funciona como un módulo que se integra directamente en servidores web como Apache o Nginx, permitiendo un análisis y un filtrado del tráfico HTTP en tiempo real. Su arquitectura se basa en procesar las solicitudes y responder las solicitudes HTTP a través de varias fases:

- Primero analiza los encabezados entrantes de la solicitud y examina patrones maliciosos a detectar.
- Luego analiza el cuerpo de la solicitud POST e Inspecciona otros métodos que contienen datos en el cuerpo.
- Después analiza los encabezados de respuesta o salientes y Revisa los encabezados para identificar posibles filtraciones de información.
- También analiza el cuerpo de la respuesta del servidor para detectar posibles vulnerabilidades o filtraciones de datos sensibles.
- Por ultimo, Registra detalladamente las transacciones en un log, facilitando el análisis forense y de seguridad.

ModSecurity usa un lenguaje de reglas propio llamado SecRules, permitiendo definir acciones mas específicas para filtrar el tráfico HTTP. Además, es compatible con las reglas OWASP (CRS), proporcionando una buena base para la protección contra las vulnerabilidades más comunes.

Características generales

La primera característica a destacar es su compatibilidad multiplataforma, ya que funciona como módulo en servidores web como Apache, que es el usado en este proyecto.

Tiene un lenguaje de reglas bastante potente, usando SecRules, un lenguaje muy flexible para definir tanto reglas como políticas de seguridad personalizadas al proyecto.

Como los demás WAF vistos, ofrece la integración con OWASP CRS, dando la compatibilidad con el conjunto de reglas OWASP CRS, que dan una protección contra las amenazas más comunes muy completa. Gracias a estas reglas, se puede detectar ataques comunes como: Protección contra inyecciones SQL, XSS, inclusión de archivos, y otros ataques listados en OWASP.

Una muy buena característica es el análisis en tiempo real que proporciona, da una inspección y filtrado del tráfico HTTP en tiempo real, permitiendo respuestas inmediatas a amenazas.

Registra detalladamente el trafico recogido en un log de las transacciones. Dando soporte para decodificar la URL y eliminar caracteres evasivos. Y tiene una persistencia de datos que permite el seguimiento de sesiones y usuarios a través de variables transaccionales.

Ventajas

ModSecurity es el que mas ventajas ofrece frente a los otros vistos. Proporciona mas madurez y estabilidad frente a otras opciones, con más de dos décadas de desarrollo, ModSecurity es una solución probada y confiable.

Tiene una flexibilidad y personalización muy amplia, gracias a sus dos lenguajes de reglas permite una personalización detallada de las políticas de seguridad acorde con la aplicación web que se esta desarrollando.

Hay una amplia comunidad y documentación bastante activa, esto es muy importante a tener en cuenta ya que es de gran ayuda a la hora de encontrarte con problemas de configuración o mantenimiento.Con una comunidad activa y una extensa documentación, facilita la resolución de problemas e implementación de nuevas funcionalidades.

Lo mas llamativo en la compatibilidad con múltiples servidores web. Ya sea su buena integración con Apache o incluso Nginx que lo hace versátil para diferentes entornos y la posibilidad de implementarlo en cualquier proyecto.

Por ultimo tiene una gran capacidad de respuesta inmediata. Como se ha visto antes, al estar integrado directamente en el servidor web, puede responder rápidamente a amenazas sin necesidad de redirigir el tráfico a servicios externos, muy dinámico y por ende mejora el comprensión al usar este WAF.

Limitaciones

Como en los anteriores WAF, no todo es perfecto y surjen desventajas. Para empezar requiere un tiempo de comprensión y estudio del entorno previo. Si no tienes experiencia o es tu primer WAF la curva de aprendizaje es algo pronunciada. Requiere conocimientos avanzados para escribir y mantener reglas efectivas, lo que puede ser un desafío para equipos con menos experiencia o para un entorno pequeño con es este.

Surje cierta complejidad en la gestión de reglas y actualizarlas se puede volver bastante complejo en entornos grandes o dinámicos.

Como bien hemos visto en otros WAF, el rendimiento es importante y sobre todo en entornos de alto tráfico. El análisis detallado de las solicitudes puede impactar en el rendimiento del servidor si no se configura adecuadamente.

Puede dar falsos positivos si las reglas no se personalizan adecuadamente, ya pueden llegar a ser algo sensibles y generar falsos positivos, bloqueando tráfico legítimo si no se ajustan correctamente.

A diferencia con BunkerWeb e igual que coraza, carece de interfaz gráfica nativa. No cuenta con una interfaz gráfica integrada que haga mas visual la experiencia, lo que puede dificultar la gestión y el monitorio para algunos usuarios.

Por ultimo, tiene una dependencia con servidor web elevada. Al estar integrado como módulo, su funcionamiento está estrechamente ligado al servidor web, lo que puede limitar su flexibilidad en ciertos entornos.

Conclusión

ModSecurity es una solución bastante robusta para defender aplicaciones web, mas en detalle en entornos donde se necesite ver detalladamente que reglas de seguridad se están activando. Su integración directa con servidores web como Apache y la integración de reglas predefinidas como las de OWASP CRS lo hacen una herramienta muy potente para detectar y prohibir una amplia gama de amenazas.

Sin embargo, su dificultad para comprenderlo y la necesidad de una gestión cuidadosa de las reglas pueden no ser tan buena opción en ciertos entornos, especialmente aquellos con recursos mas limitados. En comparación con soluciones más modernas y automatizadas como BunkerWeb o Coraza, ModSecurity ofrece un mayor grado de control y personalización, pero a costa de una mayor complejidad de comprensión y aprendizaje.

En definitiva, la elección de ModSecurity como WAF dependerá de las necesidades específicas del proyecto, sobre todo en que tecnologías se basara al desarrollarlo y en la experiencia del equipo en la gestión de políticas de seguridad web.

6.2. Proceso de elección

Una vez expuestas las tres opciones consideradas para este proyecto (BunkerWeb, Coraza y ModSecurity), pasamos a detallar el proceso de elección seguido hasta llegar a la solución final. Este proceso tuvo en cuenta tanto las ventajas y desventajas de cada alternativa como la compatibilidad con las tecnologías utilizadas, además del tiempo y los recursos disponibles.

Primera opción: BunkerWeb

La primera herramienta que se evaluó fue BunkerWeb, principalmente porque no requería un conocimiento avanzado en seguridad, lo cual era ideal en este proyecto, proveniente de la mención de software. BunkerWeb ofrecía una interfaz pensada para facilitar la configuración y entendimiento del WAF, lo que lo convertía en una elección lógica para comenzar.

Se descargó la imagen Docker correspondiente y se preparó una configuración inicial en el docker-compose. No obstante, al intentar acceder a la interfaz de administración desde el navegador a través del puerto expuesto, comenzaron a surgir diversos problemas.

Para poder utilizar correctamente la interfaz, era necesario configurar con cierto detalle el archivo YAML, lo cual resultó contradictorio: se esperaba una solución visual e intuitiva, pero en la práctica, se debía configurar manualmente gran parte del sistema desde archivos YAML, sin los beneficios esperados de una interfaz gráfica.

En los intentos por acceder al panel, si la configuración no era totalmente correcta, la interfaz ni siquiera cargaba. En caso de lograr una configuración válida, el sistema quedaba bloqueado en un bucle de búsqueda de dependencias.

Se realizaron numerosos intentos, incluyendo el cambio de sistema operativo. En la documentación se recomendaba ejecutar BunkerWeb sobre distribuciones Linux. Por tanto, se realizaron pruebas en Windows 11, Ubuntu 22.04, Ubuntu 24.04, Linux Mint y otras imágenes Linux desde Docker. Sin embargo, los problemas persistieron de forma consistente en todos los entornos.

Como último intento, se regresó a Windows utilizando otra imagen de Docker, pero la más actualizada y documentada seguía siendo la que ya se había probado sin éxito. Investigando más a fondo, se observó que BunkerWeb está especialmente diseñado para integrarse con entornos profesionales y de pago como Amazon Web Services, lo cual no era viable para un proyecto académico o personal.

En resumen, BunkerWeb no era una solución adecuada para proyectos pequeños ni para entornos basados en Apache y PHP, ya que presentó numerosos problemas de compatibilidad y un consumo excesivo de tiempo en tareas de configuración.

Segunda opción: Coraza

Tras descartar BunkerWeb, se evaluó Coraza, que se instaló utilizando su imagen oficial, la cual ya incluía el servidor web Caddy. Esta vez la instalación e implementación inicial resultaron exitosas, y se integró adecuadamente al entorno del proyecto.

Sin embargo, surgió un nuevo inconveniente: el proyecto pasaba a tener tres contenedores (Coraza + Caddy, Apache + PHP y la base de datos), lo que rompía la estructura inicialmente definida de trabajar con solo dos contenedores. Aunque técnicamente era posible continuar así, implicaba una complejidad innecesaria en comparación con los beneficios reales, especialmente tratándose de un entorno académico.

Se evaluó durante varios días la posibilidad de mantener tres contenedores, pero se concluyó que se requeriría implementar medidas de seguridad adicionales para proteger la comunicación entre servicios, algo que no aportaba valor significativo al objetivo del proyecto, que era precisamente reducir riesgos mediante una arquitectura segura pero simplificada.

Posteriormente, se intentó integrar Coraza directamente con Apache en un mismo contenedor, pero no existía compatibilidad directa entre ambas tecnologías. Tras múltiples errores en la integración y sin documentación clara que facilitara esta tarea, se descartó también esta opción.

Tercera opción: ModSecurity

Como última alternativa, se evaluó ModSecurity, y el resultado fue radicalmente distinto. Gracias a su amplia documentación y al respaldo de una comunidad activa, fue posible integrarlo rápidamente y sin complicaciones junto con Apache y PHP en un entorno de solo dos contenedores, cumpliendo así los requisitos iniciales del proyecto. ModSecurity demostró ser la opción más madura y estable. Su comportamiento fue totalmente predecible, y la compatibilidad con Apache y PHP era nativa, por lo que no fue necesario realizar configuraciones complejas o forzadas.

Característica	BunkerWeb	Coraza	ModSecurity
Lenguaje / Base tecnológica	Docker + Nginx.	Go	C (nativo para Apache/Nginx)
Instalación	Moderada – GUI atractiva, pero requiere configuración YAML	Alta – fácil con Caddy.	Alta – buena documentación, integración directa
Apache + PHP	Limitada, requiere adaptaciones	Muy limitada, sin soporte oficial directo	Excelente – nativa y estable
Interfaz gráfica	Sí	No	No
Curva de aprendizaje	Baja al principio, luego alta por configuración técnica	Moderada	Alta
Soporte y comunidad	Limitado, enfocado a entornos comerciales	Comunidad activa (OWASP)	Muy amplia
OWASP CRS	Compatible	Compatible	Compatible
Entornos recomendados	Cloud profesionales (AWS)	Cloud-native (Kubernetes, Go, Caddy)	Infraestructura tradicional (Apache/Nginx)
Documentación	Escasa y poco clara	Clara, aún en evolución	Muy completa
Rendimiento	Bueno si se optimiza	Alto – desarrollado en Go	Bueno
Personalización	Limitada fuera de su GUI	Alta – admite plugins	Alta – reglas personalizadas en SecLang
Resultado en el proyecto	No funcional, problemas graves de compatibilidad	Parcialmente funcional, descartado por arquitectura	Totalmente funcional, elegido como solución final

Tabla 6.1: Comparativa final entre BunkerWeb, Coraza y ModSecurity

Reflexión final

Durante la fase inicial del proyecto, todas las opciones analizadas parecían compatibles con Apache y PHP según sus respectivas documentaciones. Sin embargo, la experiencia práctica demostró que esa compatibilidad era teórica o limitada en algunos casos, especialmente con BunkerWeb y Coraza.

En cambio, ModSecurity fue la única solución que cumplió plenamente con lo prometido en cuanto a compatibilidad, rendimiento y facilidad de integración. Las experiencias previas con BunkerWeb y Coraza, aunque no dieron los resultados esperados, sirvieron para profundizar en el funcionamiento de los WAFs, Docker y la seguridad web, facilitando luego la configuración óptima de ModSecurity.

En conclusión, aunque se invirtió un tiempo considerable en la evaluación de soluciones que finalmente no se utilizaron, este proceso fue clave para llegar a una elección informada y adecuada. ModSecurity no solo fue la solución técnica más eficaz, sino también la que mejor se adaptó al contexto del proyecto, tanto por su madurez como por su integración directa con las herramientas utilizadas.

6.3. Estructura del contenedor

Antes de ver como funciona Modsecurity por dentro, se va a abordar como se ha instalado en el contenedor. Para ello se ha usado el archivo Dockerfile segmentado a continuación:

- Instalación de PHP y Apache desde un mismo modulo. Se tiene que instalar primero, ya que es la base del contenedor.

```
FROM php:8.2-apache
```

Figura 6.2: Instalación PHP con apache

- Instalación de Modsecurity en el contenedor, con el módulo libapache2-mod-security2.

```
RUN apt-get update && apt-get install -y --no-install-recommends libapache2-mod-security2 git wget \
 && rm -rf /var/lib/apt/lists/* \
 && a2enmod security2
```

Figura 6.3: Instalación Modsecurity

- Activamos Modsecurity y cambiamos privilegios.

```
RUN cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf \
 && sed -ri 's/SecRuleEngine DetectionOnly/SecRuleEngine On/' /etc/modsecurity/modsecurity.conf \
 && echo "SecAuditEngine RelevantOnly"       >> /etc/modsecurity/modsecurity.conf \
 && echo "SecAuditLog /var/log/modsec_audit.log" >> /etc/modsecurity/modsecurity.conf \
```

Figura 6.4: Activación Modsecurity

Se copia y modifica el archivo de configuración para ponerlo en modo bloqueo y se define el archivo de los logs donde se registran los ataques. Pero surge un problema, cuando php quiera acceder a este log para mostrárselos a los administradores necesitará ser root.

```
&& touch /var/log/modsec_audit.log \
&& chmod 644 /var/log/modsec_audit.log \
&& chown www-data:www-data /var/log/modsec_audit.log
```

Figura 6.5: Permisos del log

Ahora sí, el archivo de logs donde se registran todos los ataques se puede leer sin ningún problema y mostrar con facilidad a los administradores de la web.

- Clonamos el repositorio de OWASP CRS.

```
RUN git clone --depth 1 https://github.com/coreruleset/coreruleset.git /usr/local/owasp-crs \
 && mv /usr/local/owasp-crs/crs-setup.conf.example /usr/local/owasp-crs/crs-setup.conf \
 \
 # Ajusta las acciones por defecto: que PHASE por defecto, sólo LOG
 && sed -ri \
      -e 's/SecDefaultAction "phase:1,log,pass"/SecDefaultAction "phase:1,pass,log"/' \
      -e 's/SecDefaultAction "phase:2,log,pass"/SecDefaultAction "phase:2,pass,log"/' \
      /usr/local/owasp-crs/crs-setup.conf
```

Figura 6.6: Repositorio OWASP CRS

Se clona el repositorio de owasp crs, integrando todas las reglas por defecto que tendrá el firewall y se guardarán dentro de la ruta /usr/local/owasp-crs/rules en el contenedor.

- Instalación de mysqli y pdo_mysql.

```
RUN docker-php-ext-install mysqli pdo_mysql
```

Figura 6.7: Instalación mysqli y pdo_mysql

Esto es muy importante ya que, si no php no puede hacer peticiones a la base de datos.

- Se copian los archivos necesarios del entorno al contenedor.

```
COPY config/modsecurity.conf          /etc/modsecurity/modsecurity.conf
COPY config/crs-setup.conf            /usr/local/owasp-crs/crs-setup.conf
COPY config/modsec-remove.conf        /etc/apache2/conf-enabled/modsec-remove.conf
COPY config/apache-servername.conf    /etc/apache2/conf-enabled/apache-servername.conf

# Hay que incluir las reglas personalizadas
COPY config/waf_rules/custom.conf /usr/local/owasp-crs/rules/custom.conf
```

Figura 6.8: Copia de archivos entorno ->contenedor

Esta parte es muy importante, se crea dentro del entorno la carpeta 'configý ahí se guardan los 4 archivos de la izquierda, donde pueden modificarse cómodamente desde le entorno y así cuando se bajen o suban los contenedores estén aplicados todos los cambios de configuración del firewall, reglas etc.

Una vez se tenga la imagen instalada con todas sus dependencias desde el Dockerfile. Gracias a la extensión de docker que ofrece Visual Studio podemos ver como esta estructurado Modsecurity dentro del contenedor.

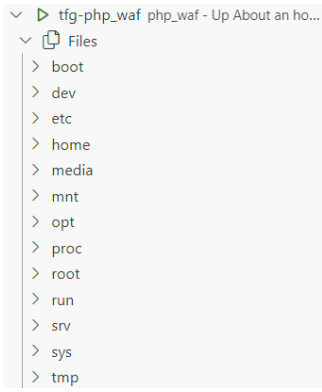

Figura 6.9: Estructura de carpetas php_waf

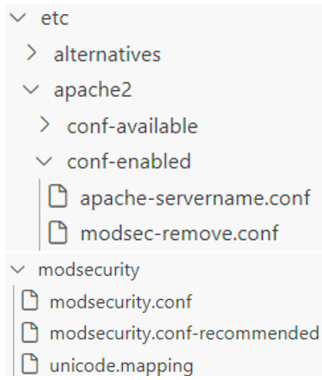

Figura 6.10: Ejemplo carpeta: /etc

Como se puede ver, aparte de seguir una estructura linux, se aprecian muchas carpetas y módulos de configuración. Como es obvio, no se tocaran todos y solo será necesaria la modificación de unos pocos. En general para configurar el WAF solo han hecho falta modificar estos archivos:

modsecurity.conf

Este archivo, ubicado en la ruta "/etc/modsecurity/modsecurity.conf" dentro del contenedor, es la configuración general del WAF. Dentro de él se encuentran 3 parámetros:

- SecRuleEngine On: Activación de ModSecurity y del motor de reglas. Hace que el WAF funcione y evalúa las reglas que se hayan definido (YA sean personalizadas o de OWASP).
- SecAuditEngine RelevantOnly: Detecta los eventos importantes y los registra en el log. Solo registra en él las reglas con acciones como "log", "deny"... Es útil para evitar llenar el log de registros innecesarios.
- SecAuditLog /var/log/modsec_audit.log: Ubicación del archivo donde se guardaran los logs. Que mas tarde es leído por un script para mostrárselo al administrador.

Esta es la configuración básica y eficiente para conseguir un WAF funcional y con un buen rendimiento.

crs-setup.conf

Este segundo archivo, ubicado dentro del contenedor en la ruta "/usr/local/owasp-crs/crs-setup.conf", no es tan relevante, ya que su objetivo es personalizar, activar o modificar comportamientos generales de OWASP.

Se podría eliminar ya que no se modifica ninguna regla de OWASP. Pero si en un futuro se quiere modificar alguna de estas reglas, siempre es bueno tenerlo a mano y no dificultar el proceso de mejorar el proyecto en el futuro.

apache-servername.conf

Este archivo, guardado en la carpeta de apache "/etc/apache2/conf-enabled", es algo necesario. Ya que dentro de él, se establece el nombre del servidor (servername offmyke) que apache usa para identificar las solicitudes de la web, cuando hay múltiples virtualHosts.

Evita que apache muestre advertencias y compile correctamente cuando se ejecuta. Se podría quitar, pero teniendo en cuenta que dentro del archivo hosts, se tiene configurado el localhost como "127.0.0.1 offmyke" es muy recomendable dejarlo para evitar problemas y que apache pueda reconocerlo.

Reglas OWASP CRS

Para ver las reglas de OWASP, compatibles con el WAF del proyecto, se debe indagar dentro de la ruta "/usr/local/owasp-crs/rules", en donde se encuentran todos los archivos con una gran variedad de reglas ya definidas.

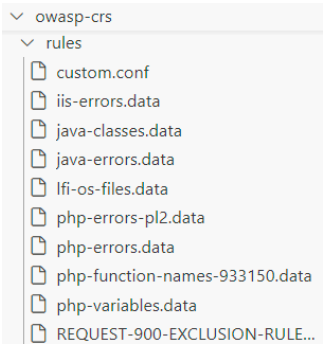

Figura 6.11: Carpeta rules de OWASP

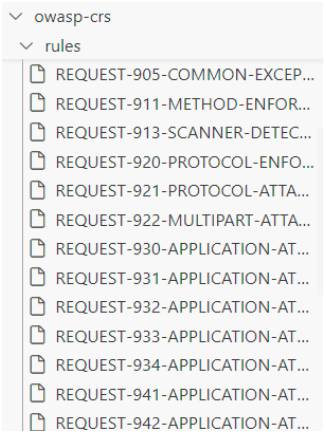

Figura 6.12: Carpeta rules de OWASP

custom.conf

En este archivo, que se encuentra en la ruta "/usr/local/owasp-crs/rules/custom.conf", guarda todas las reglas personalizadas que se han creado a mayores para mas seguridad y mas adaptabilidad al proyecto.

modsec-remove.conf

Por ultimo, hay que tener en cuenta que modsecurity en ocasiones, genera falsos positivos. Estos falsos positivos se deben ignorar o no deben meterse en el log, para que los administradores no los vean como una amenaza.

Por ejemplo, durante una batería de pruebas, cuando se introdujo el favicon.ico en la web, al hacer ciertas peticiones, se mostraba como un ataque. Esto simplemente muestra un icono en la pestaña del navegador y no se puede considerar un ataque. Por lo que en este archivo se pueden configurar reglas para evitar estos falsos positivos.

Figura 6.13: Archivo modsec-remove.conf

En resumen, la estructura del contenedor Docker en el que se ejecuta ModSecurity está diseñada de forma modular y organizada, siguiendo la jerarquía típica de un sistema Linux. Esto facilita tanto la configuración como el mantenimiento del entorno. Aunque existen numerosos archivos y carpetas dentro del contenedor, únicamente es necesario intervenir en unos pocos para garantizar un funcionamiento correcto y seguro del WAF.

Los archivos clave, como modsecurity.conf, custom.conf y modsec-remove.conf, permiten ajustar el comportamiento del firewall según las necesidades del proyecto, ya sea activando reglas personalizadas, eliminando falsos positivos o simplemente estableciendo la configuración base. Otros ficheros como apache-servername.conf o crs-setup.conf, aunque secundarios en esta implementación concreta, pueden resultar útiles en el futuro si se desea escalar o adaptar el sistema.

Además, dentro del contenedor también se encuentra la configuración esencial de PHP y del servidor Apache, lo que permite gestionar desde un solo entorno tanto la lógica del servidor como las reglas de seguridad. Esto aporta un gran control y flexibilidad para mantener el entorno seguro y adaptado a las necesidades específicas del proyecto.

Gracias a esta estructura se consigue un entorno de seguridad robusto, eficiente y fácilmente gestionable dentro de Docker.

6.4. Exposición pública mediante un túnel seguro

Esta sección se ha decidido incluir dentro de este capítulo, seguridad, ya que trata de exponer la aplicación web a otros usuarios. A términos generales, la web no solo registra ataques, sino también permite a usuarios realizar casos de uso como realizar compras.

Inicialmente, se consiguió cambiar el nombre de localhost por offmyke modificando el archivo hosts del ordenador donde se ejecuta la aplicación. Pero esto solo sirve localmente. Lo más normal es que cualquier persona pueda acceder a la web desde su propio ordenador, ya que ese es su objetivo.

Una opción fácil que se investigó inicialmente fue acceder a la aplicación desde otros dispositivos conectados a la misma red Wi-Fi, metiendo directamente la IP del ordenador y el puerto que esta escuchando, por ejemplo: "http://192.168.X.X:8080". Sin embargo, esta opción resulta incómodo, ya que saber y compartir direcciones IP largas no es ideal ni seguro.

Para solucionar esto, se optó por utilizar túneles seguros. Un túnel seguro permite exponer una web en localhost a través de una URL accesible desde cualquier navegador, de forma temporal o permanente, con seguridad y sin necesidad de configuraciones complejas en el router.

Antes de instalar ningún túnel, se instaló Chocolatey, un gestor de paquetes para Windows que permite instalar aplicaciones desde la terminal de PowerShell rápidamente. A través de Chocolatey, se instalaron tres opciones de túneles distintos que se evaluaron:

Ngrok

Ngrok fue la primera opción probada. Su uso es muy sencillo:

- choco install ngrok
- ngrok config add-authtoken $YOUR_AUTHTOKEN
- ngrok http 8080

Ngrok genera una URL temporal que permite acceder a la web desde otros dispositivos. Sin embargo, presentaba tres inconvenientes:

Las URLs eran demasiado largas, haciendo poco práctica su distribución.

Solo permitía acceder a la web desde la misma red (en la versión gratuita).

Cada vez que se iniciaba el túnel, la URL cambiaba. Esto complicaba enormemente la integración con servicios como PayPal, donde se requiere una URL fija para las rutas de éxito y cancelación de compra. Obtener una URL fija requería pagar una suscripción.

Cloudflare Tunnel

La segunda alternativa fue Cloudflare. También es gratuito, pero su uso resultó más complejo. No ofrece directamente un dominio gratuito, sino que es necesario registrar uno por separado y enlazarlo con Cloudflare, lo que implica más pasos y complicaciones con aplicaciones de terceros. Debido a esto, también fue descartado.

LocalTunnel

Finalmente, se optó por LocalTunnel, ya que resultó ser la solución más cómoda y funcional para este proyecto. Permite definir una URL personalizada desde el inicio, como:

- npx localtunnel –port 8080 –subdomain offmyke

De esta forma, se genera una URL del tipo “https://offmyke.loca.lt“, que es más fácil de recordar y compartir. Aunque al final de la URL aparece una especie de "marca de agua"(subdominio), esto no afecta al funcionamiento.

La única limitación es que LocalTunnel solo permite acceder desde la misma red local. Para hacerlo accesible desde el exterior, sería necesario abrir puertos en el router y registrar un dominio real, lo cual se consideró innecesario para el alcance del proyecto (Se podría pensar en un futuro). Con esta solución, cualquier usuario conectado a la misma red Wi-Fi puede acceder a la aplicación web de forma profesional y sin necesidad de modificar las rutas de retorno o éxito de PayPal.

Conclusión

El uso de túneles seguros permite exponer de manera temporal y controlada una aplicación local al exterior, sin realizar configuraciones complejas ni comprometer la seguridad. Tras probar varias opciones, LocalTunnel se consolidó como la alternativa más práctica para entornos de prueba o redes locales, permitiendo compartir la aplicación web de forma rápida, gratuita y funcional.

Capítulo 7

prueba practica de seguridad

El objetivo de este capítulo es evaluar y mostrar la efectividad del WAF implementado en el proyecto, en este caso ModSecurity, frente a ataques comunes. Se simularán diversos escenarios, tanto manuales como automáticos para ver como detecta, bloquea y registra los ataques.

Para realizar estas pruebas se han preparado reglas y sus correspondientes ataques para mostrar como se activan y bloquean la solicitud. Nada más bloquear el ataque, se registra toda la información tanto en un archivo csv (Excel), como en la base de datos. Todos estos datos se podrán ver desde el panel de administración de la web.

Se pueden realizar tres tipos de ataques para comprobar la protección del WAF:

- Desde el navegador, manipulando los parámetros directamente en la URL (método `GET`).
- Desde los formularios de la propia web, insertando código malicioso en los campos visibles. En el caso del proyecto, no se manipulan formularios con métodos GET.
- Usando herramientas externas como `curl` o `Postman`, para enviar peticiones `POST` personalizadas y hacer peticiones más elaboradas.

Ahora, se muestra cómo responden las reglas OWASP ante ciertos ataques comunes. Como se explicó anteriormente, estas reglas se almacenan en los directorios internos del contenedor del WAF. Demostrando su efectividad frente a diferentes ataques.

Todos estos ataques devolverán un forbidden:

- Inyección SQL `https://offmyke.loca.lt/user/register.php?username='OR'1'='1"`. Aparece diferente en el navegador por como interpreta los caracteres.

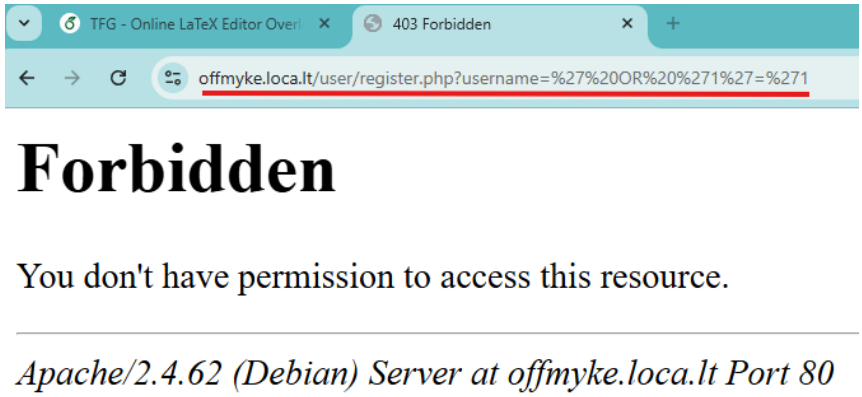

Figura 7.1: Prueba inyección SQL

- XSS `https://offmyke.loca.lt/user/login.php?email=\"<script>alert(1)</script>"`. Intenta ejecutar un script malicioso en la web.
- LFI `"https://offmyke.loca.lt/?file=../../../../etc/passwd"`. Intenta acceder a archivos de dentro del contenedor.
- RCE `"https://offmyke.loca.lt/?cmd=ls%20-al"`. Intenta ejecutar comandos en el sistema.

Ejemplo de ataque con curl:

- ATTACK-PHP curl.exe `"https://offmyke.loca.lt/?code=phpinfo()"`. Devuelve el codigo de error como los anteriores.

```
PS C:\Users\Myke\Desktop\TFG\TFG> curl.exe "https://offmyke.loca.lt/?code=phpinfo()"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.62 (Debian) Server at offmyke.loca.lt Port 80</address>
</body></html>
```

Figura 7.2: Prueba curl ATTACK-PHP

Tabla con los tipos de ataques que acabamos a ver, con su payload, tipo y resultado esperado.

Ataque	**Tipo**	**Ejemplo payload**	**Resultado**
Inyección SQL	Url Navegador	`' OR '1'='1`	Error 403
XSS	Url Navegador	`<script>alert(1)</script>`	Error 403
LFI	Url Navegador	`../../../etc/passwd`	Error 403
RCE	Url Navegador	`ls%20-la`	Error 403
ATTACK-PHP	Curl	`phpinfo()`	Error 403

Tabla 7.1: Ataques comunes detectados por reglas OWASP

Una vez lanzadas todas las solicitudes, se almacenan en la base de datos y en el archivo csv. Desde la web, si un administrador accede, podrá ver la siguiente tabla en el panel de administración:

ID	Mensaje	Id regla	IP	URL	Grave	Fecha
1	SQL Injection Attack Detected via libinjection	942100	172.18.0.1	/user/register.php?username=' OR '1'='1"	CRITICAL	05/Jun/2025:16:57:21.854476 +0000
2	XSS Attack Detected via libinjection	941100	172.18.0.1	/user/login.php?email=\"<script>alert(1) </script	CRITICAL	05/Jun/2025:16:58:13.038743 +0000
3	XSS Attack Detected via libinjection	941100	172.18.0.1	/user/login.php?email=\"<script>alert(1) </script>"	CRITICAL	05/Jun/2025:16:58:13.395921 +0000
4	Path Traversal Attack (/../) or (/..../)	930100	172.18.0.1	/?file=../../../../etc/passwd	CRITICAL	05/Jun/2025:16:58:35.717270 +0000
5	Remote Command Execution: Direct Unix Command Execution	932250	172.18.0.1	/?cmd=ls -al	CRITICAL	05/Jun/2025:16:58:54.755396 +0000
6	PHP Injection Attack: High-Risk PHP Function Name Found	933150	172.18.0.1	/?code=phpinfo()	CRITICAL	05/Jun/2025:16:59:14.951162 +0000

Figura 7.3: Registro Ataques

```
Fecha,IP,URI,ID Regla,Mensaje,Severidad
"05/Jun/2025:16:57:21.854476 +0000",172.18.0.1,/user/register.php?username=%27%20OR%20%271%27=%271%22,942100,"SQL Injection Attack Detected via libinjection",CRITICAL
"05/Jun/2025:16:58:13.038743 +0000",172.18.0.1,"/user/login.php?email=\%22%3Cscript%3Ealert(1)%3C/script",941100,"XSS Attack Detected via libinjection",CRITICAL
"05/Jun/2025:16:58:13.395921 +0000",172.18.0.1,"/user/login.php?email=\%22%3Cscript%3Ealert(1)%3C/script%3E%22",941100,"XSS Attack Detected via libinjection",CRITICAL
"05/Jun/2025:16:58:35.717270 +0000",172.18.0.1,/?file=../../../../etc/passwd,930100,"Path Traversal Attack (/../) or (/.../)",CRITICAL
"05/Jun/2025:16:58:54.755396 +0000",172.18.0.1,/?cmd=ls%20-al,932250,"Remote Command Execution: Direct Unix Command Execution",CRITICAL
"05/Jun/2025:16:59:14.951162 +0000",172.18.0.1,/?code=phpinfo(),933150,"PHP Injection Attack: High-Risk PHP Function Name Found",CRITICAL
```

Figura 7.4: CSV generado

Una vez vistos los ejemplos de como se comportan las reglas por defecto que ofrece OWASP, se va a bordar ahora algunas reglas personalizadas que se han creado para reforzar el WAF. Como ya se ha mostrado anteriormente los resultados y como se registran las amenazas. Aquí solo se mostraran que reglas se han creado:

1. Detección de herramientas de escaneo

```
SecRule REQUEST_HEADERS:User-Agent
    "(?i:(nessus|acunetix|nikto|sqlmap|nmap|masscan))" \
"phase:1,deny,log,status:403,msg:'Cliente indica uso de herramienta de
    escaneo',id:'900301'"
```

Bloquea peticiones que contengan en la cabecera de la petición un User-Agent como 'sqlmap', 'nmap'... identificando herramientas automatizadas de escaneo.

Ejemplo: curl.exe -A "sqlmap"https://offmyke.loca.lt

2. Método GET o HEAD con cuerpo no permitido

```
SecRule REQUEST_METHOD "^(?:GET|HEAD)$" "chain,phase:2,deny,log,status:403,\
msg:'GET/HEAD con cuerpo, no esta permitido',id:'900203'"
SecRule REQUEST_HEADERS:Content-Length "!^0?$" "t:none"
```

Evita el uso indebido de métodos GET o HEAD que incluyan un cuerpo (cosa que no debería ocurrir según el protocolo HTTP, nunca debería haber un cuerpo).

Ejemplo: curl.exe -X GET -d çuerpo=no-va"https://offmyke.loca.lt/

3. Cabecera Host como IP (no permitido)

```
SecRule REQUEST_HEADERS:Host "^[0-9\.]+$" "phase:1,deny,\
msg:'Host (cabecera) es IP, no permitido',id:'900206'"
```

Rechaza solicitudes donde la cabecera Host es una IP, lo cual puede indicar escaneo o acceso directo sin DNS.

Ejemplo: curl.exe -H "Host: 127.0.0.1"https://offmyke.loca.lt/

4. Método HTTP no permitido

```
SecRule REQUEST_METHOD "!^(?:GET|POST|HEAD|OPTIONS)$" "phase:1,deny,log,\
status:403,msg:'Metodo no permitido',id:'900207'"
```

Permite solo métodos comunes como GET, POST, HEAD u OPTIONS. Bloquea DELETE, PUT, PATCH, etc.

Ejemplo: curl.exe -X DELETE https://offmyke.loca.lt/

5. Validación de formato de email

```
SecRule ARGS:email "!@rx ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$" \
"phase:2,deny,log,status:403,msg:'Formato de email invalido',id:900601"
```

Rechaza emails que no sigan un formato válido.

Ejemplo: `https://offmyke.loca.lt/user/login.php?email=no-valido`

6. Validación de nombre de usuario

```
SecRule ARGS:username "!@rx ^[A-Za-z\s]{3,20}$" \
"phase:2,deny,log,status:403,msg:'Username invalido (solo letras y espacios, 3-20
    caracteres)',id:900603"
```

Restringe nombres de usuario a solo letras y espacios (de 3 a 20 caracteres), evitando cualquier intento de inyección de ataque.

Ejemplo: `https://offmyke.loca.lt/user/login.php?username=123@#$`

7. Evasión por codificación URL inválida

```
SecRule REQUEST_URI|ARGS|ARGS_NAMES|REQUEST_HEADERS "@validateUrlEncoding" \
"phase:2,deny,log,status:403,msg:'Evasion: codificacion URL invalida
    detectada',id:'900001'"
```

Detecta codificaciones mal formadas para intentar saltarse el firewall.

Ejemplo: `https://offmyke.loca.lt/user/login.php?username=%E0%A4%A`

8. Evasión por codificación UTF-8 inválida

```
SecRule REQUEST_URI|ARGS|ARGS_NAMES|REQUEST_HEADERS "@validateUtf8Encoding" \
"phase:2,deny,log,status:403,msg:'Evasion: codificacion UTF-8 invalida
    detectada',id:'900002'"
```

Evita ataques por uso de UTF-8 mal formado para esconder payloads.

Ejemplo: `https://offmyke.loca.lt/user/login.php?username=%C3%28`

9. Acceso a rutas sensibles

```
SecRule REQUEST_URI "@rx
    \.(phpinfo|shell|git|env|sql|conf|csv|yml|json|log|lock|html.php|db.php)" \
"phase:1,deny,log,status:403,msg:'Acceso a ruta sensible detectado',id:'900303'"
```

Bloquea intentos de acceder a archivos de configuración, código o datos confidenciales del proyecto internos, que no se quieran poner públicos.

Ejemplo: `https://offmyke.loca.lt/paypal/paypal.env`

10. Demasiados parámetros en la petición

```
SecRule &ARGS "@gt 5" "phase:2,deny,log,status:403,msg:'Demasiados parametros en la
    peticion',id:'900401'"
```

Controla peticiones con un número anormal o exagerado de parámetros, en esta regla solo deja poner como máximo 5 (posible evasión o ataque de fuzzing).

Ejemplo: `https://offmyke.loca.lt/user/prendaview.php?id=1&id=2&id=3&id=4&id=5&id=6`

11. Intento de SQL Injection (SELECT FROM)

```
SecRule ARGS:query "@rx (?i)\bselect\s*\*\s*from\b" \
"id:900700,phase:2,deny,log,status:403,msg:'Bloqueo: intento de SELECT FROM
    detectado'"
```

Detecta y bloquea intentos simples de SQL Injection con SELECT * FROM. Esta regla es necesario ya que investigando se encontró que ModSecurity no tiene como peligroso un SELECT. Con esta regla evitamos que accedan a registros de la base de datos.

Ejemplo: `https://offmyke.loca.lt/?query=SELECT*FROMusuarios`

Resumen

Estas son muchas de las reglas que se han implementado adicionalmente que las de OWASP, para una mayor seguridad. Se pueden añadir muchas más, dependiendo de las necesidades que se requiera. En cada una de las reglas se debe poner la condición y que hacer cuando se activan. En este caso:

- **phase**: Define en qué fase del procesamiento se evalúa la regla.
  - **1**: Análisis preliminar, antes de que la petición llegue a la aplicación. Solo se examinan cabeceras y metadatos.
  - **2**: Análisis profundo, tras recibir toda la petición. Se analizan parámetros, cuerpo y contenido completo.
- **deny**: Indica que la petición debe ser denegada si se cumple la condición de la regla.
- **log**: Hace que se registre el evento en el log cuando se activa la regla. Si la regla no es importante o es falso positivo no es necesario.
- **status:403**: Especifica el código HTTP que se devuelve al cliente (en este caso, 403 Prohibido).
- **msg**: Mensaje descriptivo que se guarda en el log para identificar el motivo del bloqueo.
- **id**: Identificador único de la regla, útil para su gestión, mantenimiento y depuración.

Capítulo 8

Conclusiones

Este Trabajo de Fin de Grado ha tenido como objetivo principal la comparación de diferentes WAFs (Web Application Firewalls) de código abierto para defender una web desarrollada y diseñada por el propio alumno. A lo largo del proyecto, se han investigado y estudiado conceptos importantes de seguridad web, se han visto tipos de vulnerabilidades y ataques comunes que se han probado en un entorno controlado. Usando diferentes herramientas y soluciones.

Entre las opciones analizadas, están *ModSecurity*, *Coraza* y *BunkerWeb*, todos estos WAF tienen enfoques y propiedades diferentes. Tras hacer pruebas y configuraciones, se ha demostrado que no cualquier WAF sirve para cualquier entorno. Finalmente con ModSecurity se hicieron pruebas de bastionado y ataques simulados comunes y se desmostó que este WAF tenía todo lo necesario como solución para el proyecto: facilidad de configuración, buen rendimiento, integración con otras herramientas y fácil implementación de nuevas reglas de seguridad que las que incluye OWASP.

El uso de contenedores Docker y túneles como *Ngrok*, *Cloudflare Tunnel* y *LocalTunnel* ha permitido un entorno flexible para hacer pruebas controladas, siendo más fácil su desarrollo y despliegue.

Además, el proyecto ha servido para aplicar lo aprendido durante la carrera, relacionados con desarrollo de software, redes, ciberseguridad, y metodologías de trabajo.

En definitiva, se ha conseguido demostrar la utilidad de usar un WAF open source (Código abierto) para mejorar la seguridad de una aplicación web real, así como la importancia de diseñar medidas de protección desde fases tempranas del desarrollo.

8.1. Líneas de trabajo futuras

Aunque se han conseguido los objetivos previstos, existen varias opciones de mejora y ampliación que deben implantarse en el futuro:

- **Mejorar pasarela de pagos**: Durante el proyecto se ha usado un entorno sandbox para desarrolladores. En el futuro debería usarse la API funcional real.
- **Dominio Real**: Como es obvio, se ha usado un túnel como solución local para presentar el proyecto. En el futuro es necesario comprar un dominio para que cualquier usuario pueda acceder a la web desde cualquier lugar.
- **Automatización de la seguridad**: Integrar herramientas para aplicar y validar automáticamente reglas WAF en cada despliegue de la aplicación.
- **Configurar a fondo ModSecurity**: Aunque se ha realizado una configuración básica de ModSecurity, queda pendiente una personalización más avanzada de sus reglas. Esto permitiría mejorar la detección de amenazas sin aumentar los falsos positivos.

Estas propuestas permitirían seguir evolucionando la seguridad de aplicaciones web en entornos reales, y aportar más valor a futuros desarrollos dentro del ámbito profesional.

Bibliografía

[1] Amazon Web Services, Inc. Aws waf - web application firewall. https://docs.aws.amazon.com/waf/latest/developerguide/. Accessed: 2025-05-04.

[2] Apache Software Foundation. Apache http server documentation. https://httpd.apache.org/docs/. Accessed: 2025-05-03.

[3] BunkerWeb. Official bunkerweb documentation. https://docs.bunkerweb.io/latest/. Accessed: 2025-03-15.

[4] Caddy Server Team. Caddy server documentation. https://caddyserver.com/docs/. Accessed: 2025-04-28.

[5] Cloudflare Inc. Cloudflare tunnel documentation. https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/. Accessed: 2025-05-15.

[6] Cloudflare Inc. Cloudflare web application firewall (waf). https://developers.cloudflare.com/waf/. Accessed: 2025-04-26.

[7] Coraza WAF Team. Coraza waf - open source web application firewall. https://coraza.io/. Accessed: 2025-04-13.

[8] CreditDonkey. Square vs stripe vs paypal: The full 2023 comparison. https://www.creditdonkey.com/square-stripe-paypal.html. Accessed: 2025-06-14.

[9] Diseño Web Pamplona. La mejor pasarela de pago para tu wordpress en 2024: Redsys o stripe. https://webpamplona.com/la-mejor-pasarela-de-pago-para-tu-wordpress-en-2024-redsys-o-stripe. Accessed: 2025-06-14.

[10] Docker, Inc. Docker documentation. https://docs.docker.com/. Accessed: 2025-06-14.

[11] Docker, Inc. Install docker engine. https://docs.docker.com/engine/install/. Accessed: 2025-06-14.

[12] Forbes Advisor. Paypal vs. stripe comparison 2025. https://www.forbes.com/advisor/business/software/paypal-vs-stripe/. Accessed: 2025-06-14.

[13] Google Cloud. Cloud armor - google cloud's web application firewall. https://cloud.google.com/armor/docs. Accessed: 2025-05-04.

[14] Gordon Lyon (Fyodor). Nmap network scanning documentation. https://nmap.org/book/man.html. Accessed: 2025-05-18.

[15] HowtoForge. Install modsecurity with apache in a docker container. https://www.howtoforge.com/install-modsecurity-with-apache-in-a-docker-container/. Accessed: 2025-04-25.

[16] Thinksys Inc. Docker best practices 2025 [updated]. https://thinksys.com/devops/docker-best-practices/. Accessed: 2025-06-14.

[17] Prachi Kothiyal. Modern docker best practices for 2025. https://talent500.com/blog/modern-docker-best-practices-2025/. Accessed: 2025-06-14.

[18] LocalTunnel Community. Localtunnel - expose localhost to the world for easy testing. https://github.com/localtunnel/localtunnel. Accessed: 2025-05-15.

[19] Mantpress. Stripe vs redsys, diferencias entre las pasarelas de pago. https://mantpress.com/blog/stripe-redsys-diferencias/. Accessed: 2025-06-14.

[20] Mantpress. Stripe vs redsys, diferencias entre las pasarelas de pago. https://mantpress.com/blog/stripe-redsys-diferencias/. Accessed: 2025-06-14.

[21] Microsoft Corporation. Visual studio 2022 v17.14 release notes. https://learn.microsoft.com/visualstudio/releases/2022/release-history. Accessed: 2025-06-14.

[22] Microsoft Corporation. Visual studio documentation. https://learn.microsoft.com/visualstudio/. Accessed: 2025-06-14.

[23] Microsoft Corporation. Web application firewall on azure application gateway. https://learn.microsoft.com/en-us/azure/web-application-firewall/. Accessed: 2025-05-04.

[24] Ngrok Inc. Ngrok documentation. https://ngrok.com/docs. Accessed: 2025-05-15.

[25] Oracle Corporation. Mysql 8.0 reference manual. https://dev.mysql.com/doc/refman/8.0/en/. Accessed: 2025-06-14.

[26] Oracle Corporation. Mysql documentation. https://dev.mysql.com/doc/. Accessed: 2025-05-20.

[27] Oracle Corporation. Mysql workbench. *Wikipedia*, 2025. Accessed: 2025-06-14.

[28] OWASP Foundation. Modsecurity v3 reference manual. https://github.com/owasp-modsecurity/ModSecurity/wiki/Reference-Manual-%28v3.x%29. Accessed: 2025-04-25.

[29] OWASP Foundation. Owasp top ten web application security risks. https://owasp.org/www-project-top-ten/, 2023. Accessed: 2025-04-20.

[30] PayPal, Inc. Paypal developer documentation. https://developer.paypal.com/docs/. Accessed: 2025-06-14.

[31] Christian Scano, Giuseppe Floris, Biagio Montaruli, and et al. Modsec-learn: Boosting modsecurity with machine learning. *arXiv preprint*, 2024. Accessed: 2025-06-14.

[32] Scrum Guides. The scrum guide. `https://scrumguides.org/scrum-guide.html`. Accessed: 2025-06-14.

[33] StackShare. Eclipse vs netbeans vs visual studio. `https://stackshare.io/stackups/eclipse-vs-netbeans-vs-visual-studio`. Accessed: 2025-06-04.

[34] TechGig. Maximize your docker efficiency: Strategies for 2025. `https://content.techgig.com/.../maximise-your-docker-efficiency-strategies-for-2025/`. Accessed: 2025-06-14.

[35] Universidad de Valladolid. Proyecto docente del trabajo de fin de grado 2019-2020 (Mención Ingeniería de Software). `https://alojamientos.uva.es/guia_docente/uploads/2019/545/46976/1/Documento.pdf`. Accessed: 2019-7-6.

[36] Zapier. Stripe vs. square: Which is best? [2025]. `https://zapier.com/blog/stripe-vs-square/`. Accessed: 2025-06-14.

Apéndice A

Manuales

A.1. Manual de despliegue e instalación

Para de poner en marcha la aplicación, es necesario cumplir con unos requisitos previos e instalar las herramientas necesarias. Entre ellas se encuentran: Visual Studio Code, PHP y Docker Desktop. Además, será necesario clonar el repositorio del proyecto, que está guardado en GitLab.

Una vez clonado el repositorio, se tendrá todo el código y archivos necesarios para el despliegue del proyecto. El contenedor principal se construye a partir de un Dockerfile personalizado, donde se instalan tanto PHP como Apache y ModSecurity. Aunque pueda parecer redundante tener PHP instalado en el equipo local y en el contenedor, esto es necesario tanto para el desarrollo en local como para el entorno dentro de Docker.

Desde el docker compose YAML, se instala automáticamente una imagen de MySQL:8, por lo que no será necesario instalar una base de datos por separado, ya se instala directamente en el contenedor.

Pasos para la instalación

- Desde Visual Studio Code, clonar el repositorio del proyecto.
- Desde la raíz del proyecto, abrir una terminal y ejecutar: `docker compose build` (Genera la imagen a partir del Dockerfile).
- Iniciar los contenedores: `docker compose up -d`

Con esto, los contenedores deberían estar corriendo y el entorno funcionando. Pero, en este punto solo sería accesible desde direcciones como `localhost`, el `servername` o la IP local. Para facilitar el acceso externo, desde una red local, se recomienda el uso de túneles seguros.

Creación de URL pública con LocalTunnel

Para exponer la aplicación web de forma segura, se ha optado por LocalTunnel. Esta herramienta permite crear una URL pública de manera sencilla que redirige hacia el entorno local. Si ya se tiene instalado Node.js, no es necesario hacer ninguna instalación. Solo se debe ejecutar:

- `npx localtunnel -port 8080 -subdomain offmyke`

Esto generará una URL del tipo: `https://offmyke.loca.lt`, a través de donde otros usuarios podrán acceder a la web. Aunque el rendimiento puede ser algo menor como una conexión directa, permite tener un certificado SSL gratis y acceso remoto sin configurar el router.

Instalación del módulo de pagos con PayPal

Para habilitar la pasarela de pagos de PayPal, es necesario instalar Composer, un gestor de dependencias para PHP. Una vez instalado, se deben ejecutar los siguientes comandos en la raíz del proyecto:

- `composer install`
- `composer init` (si no existe un `composer.json`)
- `composer require paypal/paypal-checkout-sdk`

Esto generará la carpeta `vendor/`, donde se instalarán los paquetes y archivos necesarios para usar PayPal.

Configuración de credenciales PayPal

Finalmente, será necesario configurar el archivo `paypal.env`, que está dentro de la carpeta PayPal de la carpeta `user/`. En él, se deben establecer los valores correctos para las variables de entorno:

- `PAYPAL_CLIENT_ID`
- `PAYPAL_CLIENT_SECRET`

Estas credenciales se obtienen desde la cuenta sandbox de PayPal. Se recomienda usar un usuario de tipo personal (no business) para realizar compras en la pasarela.

A.2. Manual de mantenimiento

El mantenimiento de cualquier aplicación web es fundamental para correcto funcionamiento a lo largo del tiempo. Es necesario que la web se adapte a nuevas necesidades y siempre sea segura evitando que surjan nuevas vulnerabilidades. Ahora, se detallan tareas principales de mantenimiento que son necesarias:

Mantenimiento de contenidos

El aspecto más importante, que al fin y al cabo, mantiene la web, es la actualización de prendas. Es importante añadir prendas con cierto tiempo para llamar la atención de los usuarios y así fomentar que compren.

También es importante modificar o eliminar productos que ya no estén disponibles. Actualizar imágenes, descripciones o precios. Estas tareas se pueden realizar a través de operaciones en la base de datos.

Mantenimiento de base de datos

La base de datos puede requerir mantenimiento como la revisión de errores o advertencias.

Optimizar tablas o registros inactivos que puedan no utilizarse con frecuencia. O Crear copias de seguridad para evitar perder información.

Mantenimiento del entorno

El entorno de desarrollo también puede requerir atención. Será necesario actualizar contenedores Docker si hay nuevas versiones de PHP, Apache o ModSecurity.

Actualizar el Composer de PayPal, para garantizar seguridad y compatibilidad en cualquier momento. Y finalmente renovar certificados o las credenciales de PayPal.

Revisión de Seguridad

Al ser una aplicación expuesta a internet y que puede recibir ataques, es importante revisar los logs de ModSecurity regularmente para detectar nuevos ataques. Actualizar las reglas OWASP CRS si se lanza una nueva versión y corregir falsos positivos.

Mantenimiento del código

También se recomienda mantener el código en buen estado, es decir, hay que revisar el código de forma periódica para controlar versiones mediante Git, documentar los cambios. y probarlos para ver su correcto funcionamiento.

A.3. Manual de usuario

En este manual se mostrará como los usuarios finales harán uso de la web, explicando las funciones principales de forma visual y sencilla. Para empezar el usuario tendrá que acceder a la página principal de la web `https://offmyke.loca.lt`. Mostrandole lo siguiente:

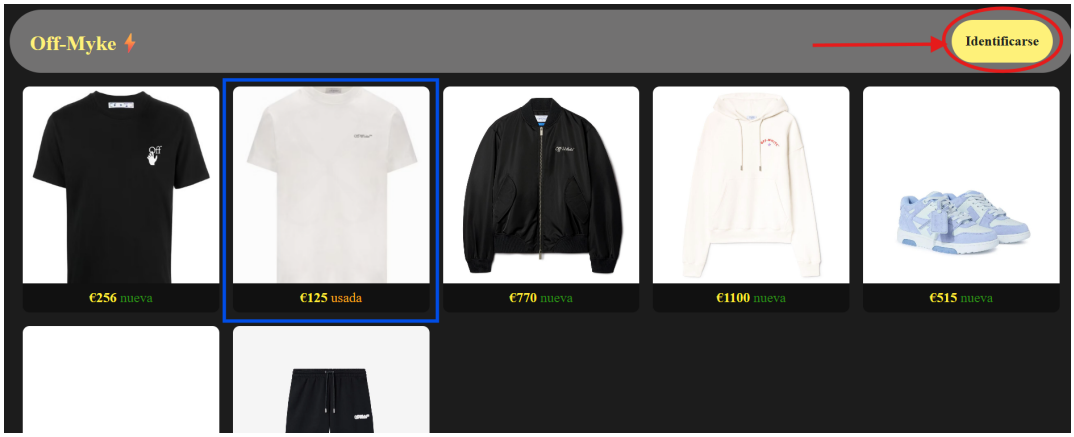

Figura A.1: Pagina principal sin registro

Una vez en la pagina principal, se pueden visualizar prendas (Cuadrado azul) o identificarse (Flecha roja). Una vez vistas las prendas se procede a la identificación:

Figura A.2: Pagina login

Si se tiene ya una cuenta registrada, se procede al login, si no, se da la oportunidad de registro. En todas las paginas de la web se ofrece la posibilidad de volver atrás, como se puede ver en la fecha roja.

Figura A.3: Pagina Register

Aquí el usuario introducirá sus datos para el registro de su nueva cuenta y si todo es correcto, le redirigirá a la pagina de login donde ahí se le solicitaran sus credenciales recién registradas.

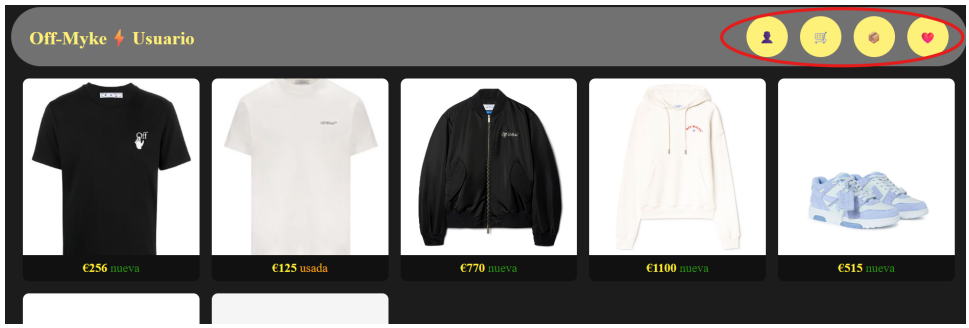

Figura A.4: Pagina principal identificado

Una vez ya identificado el usuario, se ve su nombre junto al logo y aparecen 4 funcionalidades nuevas. La primera son los datos del usuario con posibilidad de cambiarlos. El carrito donde se guardan las prendas que se quieran comprar. Los pedidos realizados por el usuario. Y por último los artículos favoritos que se quieran guardar. Si se hacen estas funcionalidades, visualmente se verán así:

Figura A.5: Botones de la web

Si queremos añadir prendas al carrito o a favoritos, se hace mediante la visualización de las prendas, con los siguientes dos botones:

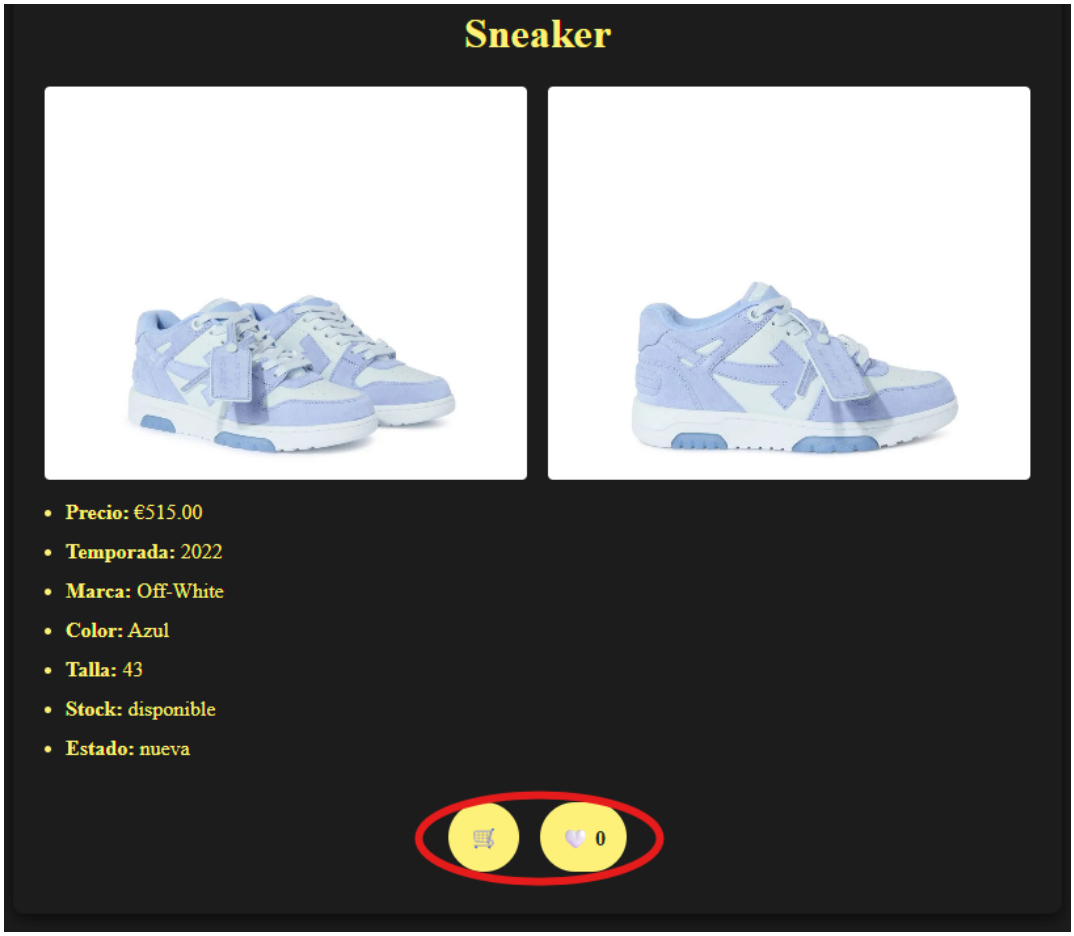

Figura A.6: Visualización de prenda

Pagina donde se pueden observar los artículos favoritos guardados, con la posibilidad de ver o eliminar según quiera el usuario:

Figura A.7: Pagina favoritos

Pagina donde se pueden observar los artículos guardados en el carrito, con la posibilidad de comprarlos todos a la vez:

Figura A.8: Pagina Carrito

Si se decide comprar los productos del carrito, se redirigirá al usurario a la pasarela de pagos:

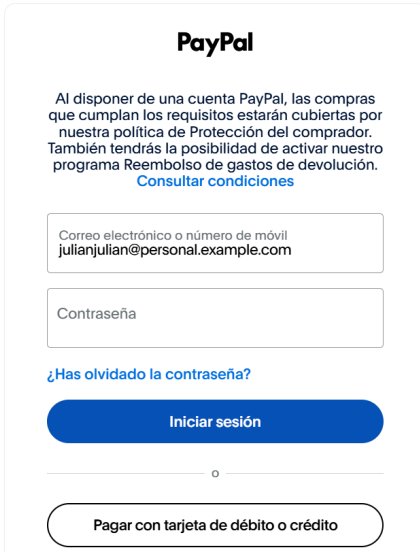

Figura A.9: Pagina PayPal Opción Tarjeta

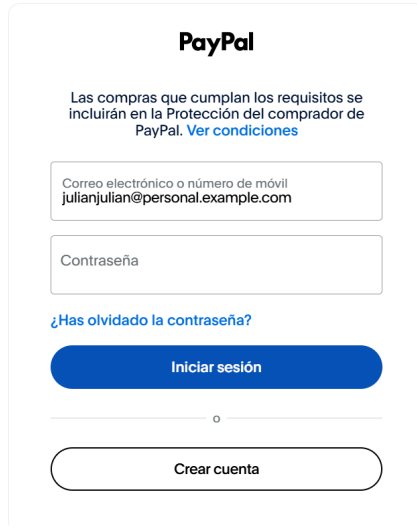

Figura A.10: Pagina PayPal Opción Registrarse

Cuando se accede a la pagina de PayPal da muchas opciones de registro. Se puede pagar con una tarjeta sin registrarse, registrarse o iniciar sesión.

Una vez ya elegida una opción, ofrece la oportunidad de pagar:

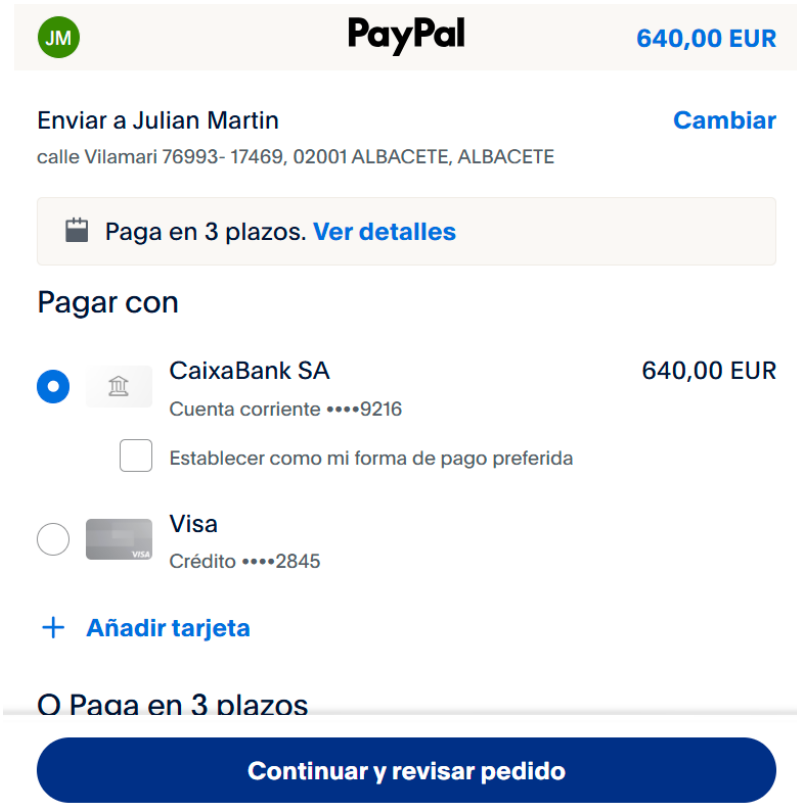

Figura A.11: Pago PayPal

Una vez que el pago es exitoso, redirige al usuario a la pagina de pedidos en donde puede, ver las prendas del pedido realizado o valorar el propio pedido.

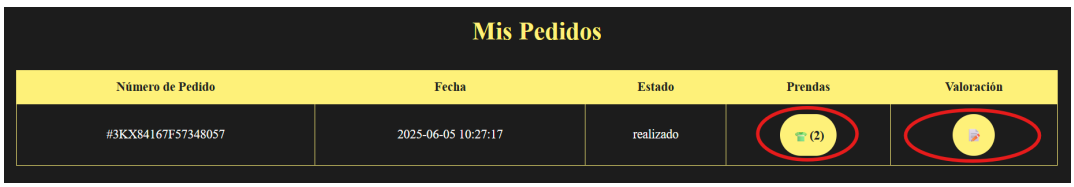

Figura A.12: Pagina pedidos realizados

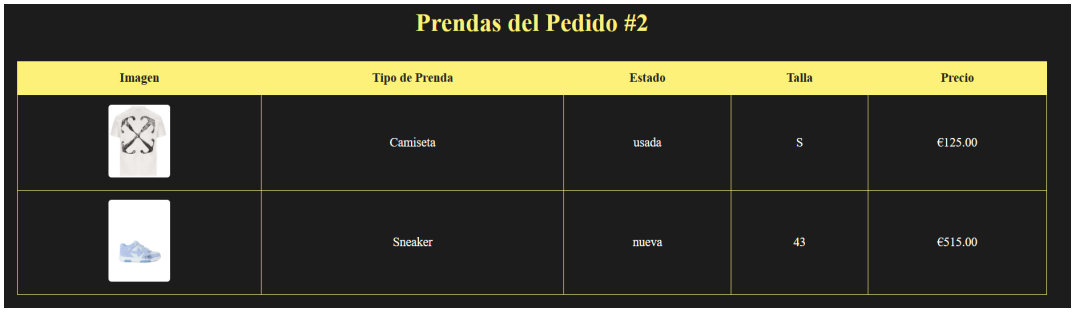

Figura A.13: Pagina ver prendas pedido

Figura A.14: Pagina valorar

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son, para Windows 11:

- Repositorio del código gitLab: `https://gitlab.com/uva9174614/TFG.git`.
- Url del sistema desplegado con LocalTunnel: `https://offmyke.loca.lt`.
- Url para descargar composer: `https://getcomposer.org/Composer-Setup.exe`.

Enlaces para descargar software utilizado en el proyecto:

- Visual Studio Code: `https://code.visualstudio.com/download`.
- Docker Desktop: `https://docs.docker.com/desktop/setup/install/windows-install/`.
- PHP versión 8.3: `https://windows.php.net/download#php-8.3`.
- Imagen de mysql:8: `https://hub.docker.com/_/mysql`.
- Repositorio de mysql:8: `https://github.com/docker-library/mysql/tree/94583e54d3bc02af523af720fdd58f8215287da9/8.0`.
- Repositorio de OWASP CRS: `https://github.com/coreruleset/coreruleset.git`.