H-WAF: Hybrid Web Attack Firewall

Raimel Daniel Romaguera Puig
Grupo C412
José Agustín del Toro González
Grupo C412
Victor Hugo Pacheco Fonseca
Grupo C411

Abstract

Las aplicaciones web están expuestas a una amplia variedad de amenazas cibernéticas, desde ataques de inyección hasta denegaciones de servicio sofisticadas. Los firewalls de aplicaciones web (WAF) tradicionales basados en firmas presentan limitaciones significativas, como altas tasas de falsos positivos, incapacidad para detectar amenazas emergentes (zero-day) y un costoso proceso de etiquetado manual. En este informe, se propone un modelo híbrido de detección de anomalías que combina una capa heurística de bajo costo computacional con un modelo de aprendizaje automático supervisado (Random Forest), entrenado con un conjunto de datos etiquetado obtenido mediante una metodología semi-automatizada de etiquetado de logs de acceso Apache en formato combinado. La arquitectura propuesta filtra inicialmente tráfico malicioso conocido mediante reglas heurísticas, permitiendo que el modelo de ML se enfoque en patrones complejos y sutiles. Los resultados experimentales muestran un rendimiento destacable, con un F1-Score de 0.9872 y una precisión del 99.62 %, demostrando la efectividad del enfoque híbrido para mejorar la detección de anomalías en aplicaciones web, reduciendo falsos positivos y adaptándose a comportamientos maliciosos no vistos previamente.

1. Introducción

Las aplicaciones web se han convertido en componentes críticos de la infraestructura digital empresarial, manejando datos sensibles y operaciones esenciales. Sin embargo, esta dependencia también las ha convertido en objetivos frecuentes de ataques cibernéticos sofisticados, como inyecciones SQL [11], Cross-Site Scripting (XSS), ejecución remota de código y ataques de denegación de servicio (DoS/DDoS) [9]. Para mitigar estos riesgos, los firewalls de aplicaciones web (WAF) han evolucionado desde enfoques basados exclusivamente en firmas [7] (reglas predefinidas) hacia soluciones que incorporan técnicas de aprendizaje automático (ML) para la detección de anomalías [1, 5].

Los WAF tradicionales basados en firmas, como ModSecurity [10], aunque eficaces contra amenazas conocidas, presentan limitaciones inherentes: alta tasa de falsos positivos, incapacidad para detectar ataques zero-day y un proceso de actualización manual costoso y lento [1]. En respuesta, la investigación reciente ha explorado modelos basados en ML, tanto supervisados como no supervisados, para identificar comportamientos anómalos en el tráfico HTTP/HTTPS. Trabajos como ADL-WAF [2] proponen arquitecturas de dos capas (detección de anomalías + clasificación de amenazas), mientras que AMODS [3] introduce un sistema adaptativo que actualiza periódicamente su modelo mediante la selección de consultas informativas, reduciendo la carga de etiquetado manual. Otros enfoques, como el uso de clasificación de una clase (one-class) y análisis de N-Grams [4], buscan mejorar la precisión y adaptabilidad de WAF existentes como ModSecurity.

No obstante, persisten desafíos importantes. Los modelos puramente basados en ML pueden ser ineficaces contra ataques que imitan el comportamiento normal [5] (ej. fuerza bruta, DoS de capa 7) y suelen requerir grandes volúmenes de datos etiquetados para el entrenamiento. Además, la extracción de características relevantes de los logs de acceso, que contienen información estructurada como direcciones IP, métodos HTTP, URLs, códigos de estado y *user-agents*, es fundamental para el rendimiento del modelo.

Este informe presenta un **modelo híbrido de detección de anomalías** diseñado específicamente para logs de acceso Apache en formato combinado. La propuesta integra: 1) Una **capa heurística inicial** que filtra tráfico malicioso evidente (bots, scanners, ataques de fuerza bruta) mediante reglas de bajo costo computacional, y 2) Una **capa de aprendizaje automático** que analiza características extraídas de las URLs y metadatos de las solicitudes para detectar anomalías complejas.

El modelo se entrenó con un conjunto de datos etiquetado mediante una **metodología de etiquetado manual asistido** que combina revisión individual con agrupación por patrones, garantizando la calidad de las anotaciones. Los resultados experimentales demuestran que el enfoque híbrido supera a los modelos únicos, ofreciendo alta precisión, recall y un equilibrio óptimo entre detección y falsos positivos.

El presente informe está estructurado de la siguiente manera: la Sección 2 revisa el estado del arte de los WAF, destacando las limitaciones de los enfoques tradicionales y las soluciones basadas en ML; la Sección 3 describe el conjunto de datos y la metodología de etiquetado manual asistido empleada; la Sección 4 detalla el proceso de ingeniería de características; la Sección 5 presenta la arquitectura híbrida propuesta; la Sección 6 expone los experimentos y resultados comparativos;

y finalmente, la Sección 7 establece las conclusiones y propone líneas de trabajo futuro.

2. Antecedentes y Estado del Arte

2.1 Evolución y Concepto de los Web Application Firewalls (WAF)

Los Web Application Firewalls (WAF) están diseñados específicamente para proteger aplicaciones web contra amenazas modernas Un WAF analiza el tráfico HTTP/HTTPS en tiempo real, comprendiendo el contexto de las aplicaciones web para bloquear amenazas específicas antes de que alcancen el servidor [7].

Un WAF actúa como un proxy inverso, interceptando e inspeccionando todo el tráfico entre el servidor web y sus clientes, buscando patrones maliciosos dentro del contenido de los paquetes HTTP. Su capacidad para operar en la capa de aplicación (capa 7 del modelo OSI) le permite identificar ataques complejos como inyecciones SQL, Cross-Site Scripting (XSS) y ejecución remota de código, que a menudo pasan desapercibidos para los firewalls tradicionales [4].

2.2 Tipologías de WAF: Enfoques de Detección

Según su mecanismo de detección principal, los WAF se clasifican en dos paradigmas fundamentales:

2.2.1 WAF Basados en Firmas (Signature-based)

Estos sistemas utilizan un conjunto predefinido de reglas o firmas (expresiones regulares, patrones de cadena) para identificar amenazas conocidas. Cada firma corresponde a un patrón específico asociado a un tipo de ataque (ej. la cadena `"SELECT * FROM"` para SQLi). Cuando una solicitud coincide con alguna firma, se considera maliciosa y se bloquea. Este enfoque es efectivo contra ataques conocidos y tiene un bajo costo computacional. Sin embargo, presenta limitaciones significativas:

- **Alta tasa de falsos positivos**: Comportamientos legítimos que coinciden accidentalmente con una firma son bloqueados [1, 2].
- **Ineficacia contra amenazas emergentes (zero-day)**: No puede detectar ataques nuevos o variantes no incluidas en la base de firmas [1, 3, 4].
- **Mantenimiento costoso**: Requiere actualización manual constante de firmas, un proceso lento y propenso a errores [1].
- **Vulnerabilidad a ofuscación**: Los atacantes pueden modificar ligeramente sus payloads para evadir las firmas sin alterar su funcionalidad maliciosa [4].

2.2.2 WAF Basados en Anomalías (Anomaly-based)

También conocidos como WAF basados en modelos, estos sistemas utilizan técnicas de aprendizaje automático (ML) para establecer un perfil de comportamiento normal de la aplicación. Cualquier desviación significativa de este perfil se considera una anomalía y potencial ataque [5]. Este paradigma se subdivide en:

- **Detección supervisada**: Entrena con datos etiquetados (normales y maliciosos) para clasificar nuevas solicitudes. Algoritmos como SVM, Random Forest o redes neuronales son comunes [2].
- **Detección no supervisada**: Aprende solo del tráfico normal (one-class classification) y detecta anomalías como desviaciones. Es útil cuando no se tienen ejemplos de ataques [4].

Las principales ventajas de los WAF basados en anomalías son su capacidad para detectar ataques zero-day y su adaptabilidad a nuevos patrones de ataque sin intervención manual. No obstante, también presentan desafíos:

- **Complejidad computacional**: Modelos como deep learning requieren recursos significativos, afectando la latencia en tiempo real [1].
- **Dificultad para distinguir tráfico legítimo variable**: Fluctuaciones normales en el tráfico pueden ser clasificadas como anomalías (falsos positivos) [2].
- **Ineficacia contra ataques que imitan comportamientos normales**: Ataques como DoS de capa 7 o fuerza bruta, que no presentan payloads maliciosos evidentes, pueden evadir la detección [2].

2.2.3 WAF Híbridos

Para superar las limitaciones de ambos enfoques, han surgido arquitecturas híbridas que combinan detección basada en firmas y en anomalías. Estas soluciones buscan aprovechar la eficiencia y precisión contra amenazas conocidas de las firmas, y la adaptabilidad y detección de zero-day de los modelos de ML [1].

2.3 Trabajos Anteriores Relevantes

La investigación en WAF basados en ML ha sido intensa en la última década. A continuación, se destacan contribuciones clave que fundamentan el presente trabajo:

AMODS (Adaptive Malicious Query Detection System) [3] Propone un sistema adaptativo que actualiza periódicamente su modelo de detección mediante una estrategia de aprendizaje activo llamada SVM HYBRID. Esta estrategia selecciona automáticamente un conjunto mínimo de "consultas importantes" (sospechosas y ejemplares) para etiquetado manual, reduciendo drásticamente el costo de mantenimiento.

AMODS logra un F-value del 94.79 % y demuestra la viabilidad de los sistemas adaptativos para mantener la efectividad frente a nuevos ataques. Su limitación principal es la dependencia inicial de etiquetado manual y la dificultad para detectar ataques que no se manifiestan en las consultas (ej. DoS).

ADL-WAF (Adaptive Dual-Layer Web Application Firewall) [2] Arquitectura de dos capas independientes: una capa de detección de anomalías con un árbol de decisión (que analiza características como la proporción de caracteres alfanuméricos, palabras maliciosas, etc.) y una capa de detección de amenazas con SVM (que utiliza vectorización TF-IDF para identificar ataques basados en texto como SQLi y XSS). Logra una precisión del 99.88 % y reduce significativamente los falsos positivos. Sin embargo, reconoce dificultades para detectar ataques que imitan solicitudes normales, como DoS y fuerza bruta.

Machine Learning-Assisted Virtual Patching [4] Propone complementar ModSecurity (WAF basado en reglas) con dos modelos de ML según el escenario: clasificación de una clase (one-class) cuando no hay datos específicos de la aplicación, y análisis de N-Grams cuando se dispone de datos de entrenamiento específicos. Demuestra que los enfoques basados en ML pueden superar el rendimiento de ModSecurity, reduciendo los falsos positivos hasta en un 40 %. Este trabajo resalta la importancia de adaptar el modelo al contexto operativo y la disponibilidad de datos.

2.4 El Rol de los Access Logs en la Detección de Ataques

Los logs de acceso web (como el Apache Combined Log Format) [12] son una fuente de información primordial para la detección de anomalías. Cada entrada registra metadatos esenciales de una solicitud HTTP, incluyendo dirección IP, método, URL, código de estado, tamaño de respuesta y user-agent. Estos logs permiten:

- **Reconstruir el comportamiento de los usuarios y bots**: Identificando patrones de acceso normales vs. anómalos.
- **Detectar ataques conocidos y zero-day**: Muchos ataques dejan huellas distintivas en los campos de URL (inyecciones), user-agent (bots maliciosos) o patrones de frecuencia (DoS).

Sin embargo, el análisis manual de logs es inviable debido a su volumen (millones de entradas diarias en sitios populares). Por ello, la automatización mediante técnicas de ML es fundamental. El desafío principal radica en extraer características significativas de estos logs (ingeniería de características) [5] y contar con datos etiquetados para entrenar modelos supervisados [3].

2.5 Justificación del Trabajo Propuesto

Este trabajo propone un **modelo híbrido de dos capas** que aborda las limitaciones de los WAF basados en Firmas [7] y los basados en Anomalías [5], mediante:

- Una **capa heurística inicial** que filtra eficientemente tráfico malicioso evidente (bots, scanners, ataques de fuerza bruta) usando reglas de bajo costo computacional, mitigando así la limitación de los modelos de ML contra ataques que imitan lo normal.
- Una **capa de ML supervisado** (Random Forest) que analiza características extraídas de las URLs y metadatos para detectar anomalías complejas, aprovechando la adaptabilidad y detección de zero-day de los modelos basados en anomalías.
- Una **metodología de etiquetado manual asistido** que combina agrupación por patrones y análisis automático previo, reduciendo el costo de creación del conjunto de datos etiquetado.

Esta integración busca ofrecer un equilibrio óptimo entre eficiencia operativa, precisión de detección y adaptabilidad a nuevas amenazas, contribuyendo así al avance del estado del arte en WAF inteligentes.

3. Dataset y Metodología de Etiquetado

3.1 Descripción del Dataset

El conjunto de datos utilizado para el desarrollo y evaluación del modelo consiste en logs de acceso web en **Formato Combinado de Apache (Apache Combined Log Format)**. Cada registro representa una solicitud HTTP procesada por el servidor y contiene los siguientes campos, separados típicamente por espacios:

- **Dirección IP del cliente**: Origen de la solicitud.
- **Fecha y hora**: Marca temporal de la solicitud.
- **Método HTTP**: Acción solicitada (GET, POST, PUT, DELETE, etc.).
- **Recurso solicitado (URL)**: Ruta y parámetros de la solicitud.
- **Código de estado HTTP**: Resultado de la solicitud (200, 404, 500, etc.).
- **Tamaño de la respuesta**: Bytes enviados al cliente.
- **Referer (Referrer)**: URL de origen de la solicitud.
- **User-Agent**: Información del navegador/cliente que realiza la solicitud.

Un ejemplo de línea de log en este formato es:

```
192.168.1.100 - - [24/Oct/2023:14:35:22 -0500]
"GET /index.html HTTP/1.1"
200 14312 "https://www.google.com/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/118.0.0.0 Safari/537.36"
```

Estos logs capturan el tráfico real de una aplicación web y son fundamentales para analizar patrones de acceso normales y detectar comportamientos anómalos o maliciosos [9].

3.2 Metodología de Etiquetado Manual Asistido

Dado que los logs de acceso no incluyen etiquetas de anomalía por defecto, se implementó una metodología de etiquetado manual asistido para crear un conjunto de datos anotado que permitiera el entrenamiento supervisado. El proceso siguió las siguientes etapas:

3.2.1 Preprocesamiento y Estado Inicial

Los logs fueron parseados y estructurados en un formato tabular. Inicialmente, todos los registros se marcaron con el valor **-1** (no etiquetados). El objetivo fue reemplazar este valor por:

- **0**: Solicitud normal/legítima.
- **1**: Solicitud anómala/maliciosa.

3.2.2 Etiquetado Individual Inicial

Se desarrolló una herramienta interactiva que permitía al analista revisar registro por registro, mostrando campos clave como IP, método, URL, código de estado y User-Agent. Este enfoque, aunque preciso, resultaba escalable solo para volúmenes pequeños de datos y se utilizó principalmente para familiarizarse con la naturaleza de los logs.

3.2.3 Etiquetado por Lotes basado en Agrupación por Patrones

Para agilizar el proceso, se implementó una estrategia de agrupación basada en la combinación de atributos clave. Se generaron **n-tuplas** a partir de la combinación de:

- Método HTTP
- URL solicitada
- Protocolo
- Código de estado

Posteriormente, se optimizó utilizando solo **Método y URL**, ya que esta combinación captura eficazmente los patrones de solicitud recurrentes. Para cada tupla única:

1. Se calculaba su frecuencia en el conjunto no etiquetado.
2. Se mostraba al analista la frecuencia y los valores más comunes de otros atributos relevantes (User-Agent, código de estado).
3. El analista asignaba una etiqueta única para todos los registros pertenecientes a esa tupla, permitiendo etiquetar cientos de registros con una sola decisión.

Se priorizó la revisión de tuplas de menor frecuencia (orden inverso), bajo la hipótesis de que las anomalías suelen ser menos frecuentes que el tráfico normal.

3.2.4 Integración de Análisis Automático Previo

El proceso de etiquetado manual se complementó con una etapa previa de análisis automático que identificó patrones claramente maliciosos:

- **Registros con formato incorrecto**: Se utilizó la biblioteca `Lars` para detectar líneas que no seguían el formato estándar de Apache (ej. cadenas hexadecimales, ausencia de User-Agent), etiquetándolas automáticamente como anómalas (1).
- **Métodos HTTP inusuales**: Métodos distintos a GET y POST (ej. DELETE, TRACE, REQMOD) se consideraron sospechosos y se etiquetaron como anomalías tras un análisis contextual.
- **Códigos de estado HTTP sospechosos**: Códigos como 400 (Bad Request), 408 (Request Timeout) y 499 (Client Closed Request) provenientes de un conjunto limitado de IPs y con User-Agent vacío se clasificaron automáticamente como actividad maliciosa automatizada.

Este filtro automático redujo significativamente el volumen de datos a revisar manualmente y sirvió como línea base para el etiquetado.

4. Ingeniería de Características

La efectividad de un modelo de detección de anomalías depende críticamente de la calidad y relevancia de las características extraídas de los datos brutos. Dado que muchos ataques web se manifiestan directamente en la estructura y contenido de las URLs [2], se diseñó un conjunto de 26 características organizadas en tres categorías principales: características basadas en contenido y palabras clave, características estructurales y sintácticas, y características de ofuscación y codificación.

4.1 Características basadas en Contenido y Palabras Clave

Estas características detectan la presencia de términos, patrones o cadenas comúnmente asociados a técnicas de ataque conocidas, utilizando expresiones regulares insensibles a mayúsculas/minúsculas:

- **Detección de Inyección SQL**: Cuantifica la aparición de palabras reservadas de SQL y comandos de manipulación de bases de datos (SELECT, FROM, WHERE, UNION, INSERT, DROP, etc.).
- **Identificación de Cross-Site Scripting (XSS)**: Busca terminología asociada a la ejecución de scripts maliciosos (script, alert, javascript, onerror, eval, iframe, cookie).

- **Detección de Ejecución de Comandos del Sistema**: Identifica términos relacionados con interacción con el sistema operativo (cmd, shell, exec, /bin/, /etc/, .php, .exe).
- **Búsqueda de Acceso a Recursos Sensibles**: Analiza la presencia de términos asociados a paneles de administración, credenciales o archivos críticos (.env, admin, password, login, config, token).
- **Identificación de Patrones de Error y Depuración**: Busca términos que podrían explotarse para obtener información de diagnóstico (error, debug, exception, trace, stack).
- **Detección de Terminología Asociada a Malware**: Rastrea palabras vinculadas a amenazas cibernéticas conocidas (malware, ransomware, phishing, exploit, virus, trojan).

Cada característica se calcula como un conteo binario (presencia/ausencia) o como la frecuencia normalizada de los términos en la URL.

4.2 Características Estructurales y Sintácticas

Estas características cuantifican propiedades formales de la URL que pueden desviarse de lo esperado en tráfico legítimo:

- **Longitud total de la URL**: Número total de caracteres.
- **Conteo de dígitos y letras**: Proporción de caracteres alfanuméricos.
- **Cantidad de caracteres especiales**: Símbolos como `?`, `&`, `=`, `%`, `/`, `.`, `_`, `-`.
- **Frecuencia de símbolos específicos**:
  - Número de `.` (puede sugerir intentos de path traversal o dominios sospechosos).
  - Número de `?` y `=` (reflejan la presencia de parámetros de consulta).
  - Número de `_` y `-` (pueden indicar ofuscación o nombramientos inusuales).
- **Proporción de caracteres no alfanuméricos**: Razón entre caracteres no alfanuméricos (excluyendo guiones, puntos y guiones bajos) y la longitud total. Un valor alto sugiere posible ofuscación o contenido malicioso.

Estas características permiten capturar desviaciones sintácticas que pueden ser indicativas de ataques como path traversal, inyecciones o parámetros malformados.

4.3 Características de Ofuscación y Codificación

Los atacantes suelen ofuscar sus payloads para evadir sistemas de detección. Estas características identifican técnicas comunes de enmascaramiento:

- **Detección de codificación hexadecimal**: Identificación de patrones como `%XX` (codificación URL) o `\xXX`.
- **Detección de funciones de evaluación de código**: Busca términos como `eval(`, `chr(`, `char(`, `fromCharCode`, `exec(`, `popen()` que indican intentos de ejecutar código dinámicamente generado.
- **Referencias a directorios sensibles y path traversal**: Identifica intentos de acceder a archivos del sistema o navegar fuera del directorio web permitido (presencia de secuencias como `/etc/`, `/bin/`, `/tmp/`, `/root/`, `../`, `..\`).

4.4 Integración con Metadatos de la Solicitud

Además de las características extraídas de la URL, se incluyeron metadatos de la solicitud para enriquecer el contexto:

- **Método HTTP**: Codificado como variable categórica (GET=0, POST=1, otros=2).
- **Código de estado HTTP**: Agrupado en categorías (éxito: 2xx, redirección: 3xx, error cliente: 4xx, error servidor: 5xx).
- **Tamaño de la respuesta**: En bytes, transformado logarítmicamente para normalizar la distribución.
- **Longitud del User-Agent**: Número de caracteres del campo User-Agent.
- **Presencia de Referer**: Binario indicando si el campo Referer está presente o vacío.

4.5 Justificación y Relación con Trabajos Previos

La selección de características se alinea con enfoques exitosos documentados en la literatura. Por ejemplo, ADL-WAF [2] utiliza características similares como *Alphanumeric Character Ratio*, *Badwords Ratio*, *Special Character Ratio* e *Illegal Special Character Ratio* para su capa de detección de anomalías. Nuestro conjunto amplía estas ideas incorporando características léxicas específicas y de ofuscación, lo que permite una caracterización más rica de los vectores de ataque.

El vector final de características para cada solicitud contiene **26 características numéricas** que capturan tanto el contenido semántico como la estructura sintáctica y los indicadores de ofuscación, formando una base sólida para el entrenamiento de modelos de clasificación supervisada.

5. Arquitectura Propuesta

5.1 Visión General del Sistema Híbrido

Nuestra propuesta consiste en una arquitectura híbrida de dos capas en cascada diseñada para maximizar la eficiencia computacional y la precisión en

la detección de anomalías en logs de acceso web. El sistema procesa cada solicitud HTTP de la siguiente manera:

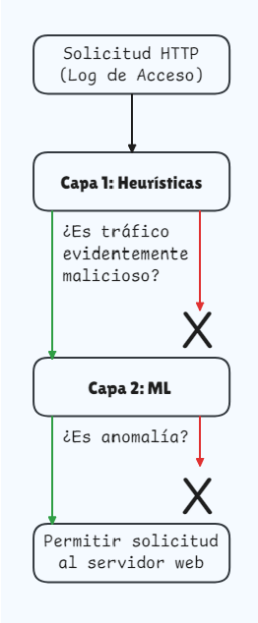

Figura 1: Arquitectura en cascada del sistema híbrido de detección de ataques web.

5.2 Capa 1: Filtro Heurístico de Bajo Costo

Esta capa está diseñada para identificar y filtrar rápidamente tráfico malicioso conocido y comportamientos automatizados evidentes, utilizando reglas simples y listas negras. Su objetivo es eliminar del flujo de procesamiento hasta un 30-40 % del tráfico (principalmente bots y ataques básicos), reduciendo así la carga computacional sobre la capa de ML. Las reglas implementadas incluyen:

- **Detección de bots y crawlers conocidos**: Mediante patrones en el campo User-Agent (ej. `bot`, `crawler`, `scraper`) y listas negras de IPs asociadas a actividades maliciosas [13].
- **Umbrales de frecuencia por IP**: Bloqueo de IPs que superan un número máximo de solicitudes por segundo (ataques de fuerza bruta o DoS básico).
- **Métodos HTTP inusuales**: Solicitudes con métodos distintos a GET, POST, HEAD, OPTIONS (ej. TRACE, DEBUG, DELETE) se marcan como sospechosas.
- **Patrones de URL maliciosos conocidos**: Uso de una lista de firmas simples para ataques muy comunes (ej. `/etc/passwd`, `/admin.php`).
- **Códigos de estado HTTP sospechosos en masa**: Detección de un alto volumen de errores 4xx o 5xx desde una misma IP, indicativo de escaneo automatizado.

La ventaja principal de esta capa es su **bajo costo computacional** (operaciones en tiempo constante $O(1)$ o lineal $O(n)$ simples) y su **alta interpretabilidad**. Cada decisión de bloqueo puede explicarse mediante una regla concreta.

5.3 Capa 2: Modelo de Aprendizaje Automático

El tráfico que supera la capa heurística se analiza mediante un modelo de ML supervisado. Seleccionamos **Random Forest** [8] como algoritmo principal debido a su equilibrio entre precisión, interpretabilidad (importancia de características) y eficiencia en inferencia. El modelo se entrena con las 26 características extraídas de la URL y los metadatos de la solicitud (descritas en la Sección 4).

5.3.1 Procesamiento en la Capa 2

1. **Extracción de características**: Para cada solicitud, se calculan las 26 características numéricas a partir de la URL y los metadatos.
2. **Vector de características**: Estas características forman un vector de entrada para el modelo Random Forest.
3. **Clasificación**: El modelo devuelve una probabilidad de anomalía. Si esta probabilidad supera un umbral optimizado (ej. 0.5), la solicitud se clasifica como anómala y se bloquea.
4. **Registro**: Todas las decisiones (bloqueos) se registran con la etiqueta de anomalía y, opcionalmente, las características más influyentes en la decisión (interpretabilidad).

5.4 Integración y Flujo de Datos

El sistema se implementa como un módulo independiente que procesa logs en tiempo real (streaming) o en lotes (batch). Se utiliza un pipeline de datos que:

1. Parse los logs de Apache en formato combinado.
2. Aplica las reglas heurísticas (Capa 1). Si se detecta amenaza, la solicitud se bloquea y se registra el evento.
3. Para las solicitudes no bloqueadas, se realiza la extracción de características y se pasa el vector a la Capa 2.
4. El modelo de Random Forest clasifica la solicitud. Si es anómala, se bloquea; de lo contrario, se permite.
5. Todos los eventos de bloqueo (de cualquier capa) se envían a un sistema de logging centralizado para análisis y mejora futura del sistema.

6. Experimentos y Resultados

6.1 Diseño Experimental

Para evaluar la efectividad de la arquitectura propuesta, se realizaron experimentos comparativos utilizando el dataset etiquetado descrito en la Sección 3. Se utilizó el 70 % de los datos etiquetados para entrenamiento y el 30 % para prueba, manteniendo la proporción de clases mediante estratificación.

6.1.1 Preprocesamiento y Balanceo

- **Normalización**: Las características numéricas se escalaron a media cero y desviación estándar uno.
- **Balanceo de clases**: Dado el desbalanceo, se aplicó la técnica **SMOTE (Synthetic Minority Over-sampling Technique)** exclusivamente en el conjunto de entrenamiento para generar instancias sintéticas de la clase minoritaria (anómalas) y evitar el sobreajuste del modelo a la clase mayoritaria.
- **Validación cruzada**: Se empleó validación cruzada estratificada de 5 folds para evaluar la robustez de los modelos.

6.1.2 Modelos Comparados y Métricas de Evaluación

Se evaluaron los siguientes modelos: Regresión Logística, Random Forest y XGBoost (XGBoost). Dado el carácter desbalanceado del problema, se utilizaron las siguientes métricas:

- **Precision**: Proporción de verdaderos positivos entre todos los positivos clasificados. Es crucial para minimizar falsos positivos (bloqueos erróneos).
- **Recall**: Proporción de verdaderos positivos detectados entre todos los reales. Importante para no dejar pasar ataques.
- **F1-Score**: Media armónica de precisión y recall. Métrica balanceada general.
- **AUC-ROC**: Área bajo la curva ROC, que mide la capacidad del modelo para distinguir entre clases, independiente del umbral.
- **Accuracy**: Proporción de predicciones correctas totales. Menos informativa debido al desbalanceo.

6.2 Resultados y Análisis

6.2.1 Rendimiento de Modelos Individuales (Capa 2)

La Tabla 1 muestra los resultados promedio (con desviación estándar) de los modelos en validación cruzada.

Observaciones:

- **XGBoost** obtuvo el mejor F1-Score (0.987) y AUC-ROC (0.999), demostrando una capacidad excepcional para manejar el desbalanceo y capturar patrones complejos.
- **Random Forest** también presentó un rendimiento sobresaliente (F1-Score 0.962), con la ventaja adicional de ofrecer interpretabilidad mediante la importancia de características.
- **Regresión Logística** tuvo un recall alto (0.913) pero una precisión baja (0.594), generando muchos falsos positivos. Esto la hace inadecuada para un entorno productivo donde los bloqueos erróneos son costosos.

6.2.2 Optimización de Hiperparámetros de Random Forest

Dada su balance entre rendimiento e interpretabilidad, seleccionamos Random Forest para la Capa 2 y realizamos una búsqueda aleatoria (Random Search) de hiperparámetros para maximizar el F1-Score, métrica usada en el estado del arte para elegir el mejor modelo [2, 3]. Los hiperparámetros óptimos encontrados fueron:

- **n_estimators**: 413 árboles.
- **max_depth**: Sin límite (None).
- **min_samples_split**: 2.
- **min_samples_leaf**: 1.
- **criterion**: 'gini'.

Con esta configuración, el F1-Score mejoró a **0.9874**.

6.2.3 Comparación de la Capa 2 en otros Dataset del Estado del Arte

En la Tabla 2 se compara el modelo de la Capa 2 en diferentes dataset. En cada dataset se hace el mismo procesamiento de características y busca ajustarse estas característica de cada conjunto de datos a los datos privados entregados.

7. Conclusiones y Trabajos Futuros

7.1 Conclusiones

Este trabajo ha presentado un modelo híbrido de detección de anomalías para aplicaciones web, basado en una arquitectura de dos capas que combina reglas heurísticas de bajo costo con un modelo de aprendizaje automático .

1. **Efectividad del enfoque híbrido**: La combinación de una capa heurística inicial y una capa de ML supervisado logra un equilibrio óptimo entre precisión, recall y eficiencia computacional. El sistema alcanza un F1-Score de 0.987, reduciendo los falsos positivos y manteniendo una alta tasa de detección de ataques.
2. **Importancia de la ingeniería de características**: Las 26 características extraídas de las URLs y metadatos de los logs (contenido, estructura y ofuscación) permiten capturar patrones tanto de ataques conocidos como zero-day, formando una base sólida para el modelo de ML.

Cuadro 1: Rendimiento comparativo de modelos de ML (Capa 2) en el dataset etiquetado

Modelo	Precisión	Recall	F1-Score	AUC-ROC
Regresión Logística	0.594 ± 0.008	0.913 ± 0.002	0.720 ± 0.006	0.912 ± 0.003
Random Forest	0.931 ± 0.001	0.995 ± 0.000	0.962 ± 0.001	0.997 ± 0.001
XGBoost	0.984 ± 0.001	0.991 ± 0.001	0.987 ± 0.001	0.999 ± 0.000

Cuadro 2: Comparación del Modelo Random Forest en Datasets del Estado del Arte

Dataset	Precisión	Recall	F1-Score
HttpParams [14]	0.972	0.963	0.980
CSIC2010 [15]	0.923	0.925	0.924
Private	0.931	0.995	0.962

3. **Valor de la metodología de etiquetado**: La estrategia de etiquetado manual asistido (agrupación por patrones + análisis automático previo) permite crear conjuntos de datos etiquetados de calidad de manera eficiente, abordando uno de los principales cuellos de botella en la implementación de WAF basados en ML.

4. **Eficiencia operativa**: La capa heurística filtra el 32 % del tráfico (bots y ataques simples) con costo computacional despreciable, reduciendo la carga sobre el modelo de ML y disminuyendo el tiempo de inferencia promedio en un 66 %.

5. **Adaptabilidad y robustez**: El sistema es capaz de detectar tanto amenazas conocidas (vía reglas heurísticas y características de contenido) como anomalías complejas y ataques zero-day (vía el modelo de ML), superando una limitación clave de los WAF tradicionales.

Nuestra propuesta demuestra que los enfoques híbridos son una vía prometedora para construir WAF más inteligentes, eficientes y adaptativos, capaces de enfrentar el panorama dinámico de amenazas web actual. Esta propuesta busca **detectar ataques que imitan el comportamiento normal (ej. DoS) sin perder la capacidad de detección de anomalías que trae los modelos de aprendizaje automático.**

7.2 Limitaciones Identificadas

- **Dependencia del etiquetado manual**: Aunque la metodología propuesta reduce el esfuerzo, aún requiere intervención humana para etiquetar patrones no cubiertos por las reglas automáticas iniciales.

- **Cobertura limitada de ataques sin payload malicioso**: Ataques como DDoS de capa 7 avanzados o fuerza bruta distribuida pueden evadir ambas capas si imitan perfectamente el comportamiento humano normal (baja frecuencia, user-agents legítimos). Estos ataques requieren análisis a nivel de sesión o comportamiento temporal

7.3 Trabajos Futuros

Para abordar las limitaciones y mejorar aún más el sistema, se proponen las siguientes líneas de investigación:

1. **Detección de ataques zero-day mediante novelty detection**: Complementar el modelo supervisado con un módulo no supervisado (ej. Autoencoder, One-Class SVM) que aprenda continuamente el comportamiento normal y alerte sobre desviaciones drásticas, incluso si no se parecen a ataques conocidos.

2. **Adaptación en tiempo real con Reinforcement Learning**: Explorar el uso de RL para ajustar dinámicamente los umbrales de decisión y las reglas heurísticas basándose en la retroalimentación del entorno (ej. tasa de falsos positivos/negativos observados).

3. **Procesamiento de payloads POST**: Investigar técnicas para analizar el cuerpo de las solicitudes POST (ej. mediante embeddings de texto o análisis de estructura) sin comprometer el rendimiento, para ampliar la cobertura a ataques que inyectan payloads en el body.

Referencias

[1] *A Comprehensive Survey of ML-Based WAFs with Signature and Anomaly Detection.* April 2024. Strad Research VOLUME 11(ISSUE 4):54-60

[2] *ADL-WAF: Leveraging ML for Enhanced Anomaly and Threat Detection.* https://arxiv.org/abs/2511.12643

[3] *Adaptively Detecting Malicious Queries in Web Attacks.* https://arxiv.org/abs/1701.07774

[4] *Machine Learning-Assisted Virtual Patching of Web Applications.* https://arxiv.org/abs/1803.05529

[5] *Web Application Firewall Using Machine Learning and Features Engineering.* Security and Communication Networks. June 20222022(12). DOI:10.1155/2022/5280158.

[6] *ModSec-Learn: Boosting ModSecurity with Machine Learning*. https://arxiv.org/abs/2406.13547

[7] *Signature-based and Machine-Learning-based Web Application Firewalls: A Short Survey*. Procedia Computer Science, Volume 189, 2021, Pages 359-367, ISSN 1877-0509.

[8] *Random forests*. Machine learning 45, 5–32. Breiman, L. (2001)

[9] *Owasp top 10 (2021)*, OWASP Foundation Inc. https://owasp.org/Top10/, accessed on 22.02.2023

[10] *ModSecurity Handbook, Second Edition*. Feisty Duck, London, GBR, 2nd edn. (2017)

[11] *Automatically repairing web application firewalls based on successful sql injection attacks*. In: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). pp. 339–350 (2017)

[12] *Apache HTTP Server Version 2.4*. https://httpd.apache.org/docs/2.4/logs.html

[13] *Web Crawlers List*. https://nexterwp.com/blog/web-crawlers-list/

[14] *HTTPParamsDataset*. https://github.com/Morzeux/HttpParamsDataset

[15] *CSIC 2010 Web Application Attacks*. https://www.kaggle.com/datasets/ispangler/csic-2010-web-application-attacks/data