OCS-WAF: un Web Application Firewall basado en anomalías con clasificadores One-Class SVM

Nico Epp
Facultad Politécnica
Universidad Nacional de Asunción
San Lorenzo, Paraguay
nicoeppfriesen@gmail.com

Ralf Funk
Facultad Politécnica
Universidad Nacional de Asunción
San Lorenzo, Paraguay
ralffunk0@gmail.com

***Resumen*—Las vulnerabilidades en aplicaciones web presentan un gran riesgo, ya que estas pueden ser explotadas por atacantes maliciosos a través de Internet. Los *Web Application Firewalls* (WAF) pueden ser colocados frente a estas aplicaciones para detectar posibles ataques y de esta forma reducir estos riesgos. En este trabajo presentamos OCS-WAF, un WAF que puede ser colocado frente a aplicaciones web para analizar los mensajes HTTP entrantes, con el fin de detectar mensajes anómalos que podrían contener ataques. Nuestra implementación, hecha en el lenguage de programación *Python*, utiliza clasificadores One-Class SVM para la detección, junto con procesos de extracción de características diseñados específicamente para mensajes HTTP. OCS-WAF es entrenado con mensajes que representan el uso normal de las aplicaciones protegidas, y posteriormente, en la fase de detección, puede detectar mensajes anómalos o ataques. Usando esta estrategia de detección de anomalías, OCS-WAF solamente necesita ser entrenado cuando haya cambios en las aplicaciones protegidas, por lo que la aparición de nuevos tipos de ataques no requiere volver a entrenarlo. Las pruebas realizadas para medir la eficacia de detección muestran que OCS-WAF alcanza un TPR promedio de $0{,}93$, un FPR promedio de $0{,}03$ y un F_1-*score* promedio de $0{,}95$ para los conjuntos de datos públicos que utilizamos. Las pruebas también evidencian que las tareas de detección de OCS-WAF no afectarían de forma notable el tiempo de respuesta de las aplicaciones protegidas. Además, puede ser entrenado con $100\,000$ mensajes normales en unos pocos minutos. Finalmente, el código fuente de OCS-WAF está disponible en un repositorio público bajo la dirección `https://github.com/nico-ralf-ii-fpuna/tfg`, con la finalidad de que otros puedan reproducir nuestros resultados y extender este trabajo en futuras investigaciones.**

***Index Terms*—Sistemas de Detección de Intrusión (IDS), Web Application Firewall (WAF), ataques web, detección de anomalías, One-Class SVM**

I. Introducción

Las aplicaciones web han tenido un gran auge en la última década, convirtiéndose en herramientas de uso masivo y frecuente para una gran cantidad de usuarios. Pero debido a que las mismas son accesibles a través de la red, están expuestas a una gran variedad de ataques [1]. Por lo tanto se necesitan soluciones para mitigar los riesgos presentes [2].

Los sistemas de detección de intrusión (IDS - *Intrusion Detection System*) son programas especializados para monitorear las actividades en un sistema o en una red en busca de intrusiones no autorizadas o posibles ataques [3]. Los IDS pueden basarse en varias fuentes de datos para sus análisis, como por ejemplo el tráfico de una red o los registros de acciones en un sistema operativo [4]. Las aplicaciones web utilizan mayormente el protocolo HTTP (*Hypertext Transfer Protocol*) [5] para sus comunicaciones; así, en el caso de un IDS que analiza mensajes HTTP, se puede hablar más específicamente de cortafuegos para aplicaciones web (WAF - *Web Application Firewall*) [4].

Los WAF pueden utilizar dos métodos distintos para la detección de intrusiones: la búsqueda de patrones de ataques conocidos, llamado también método basado en firmas de ataques (*signature-based detection*), o la búsqueda de anomalías en los mensajes HTTP con respecto al tráfico normal, ya que estas desviaciones pueden indicar ataques (*anomaly-based detection*) [4]. Para que un WAF pueda utilizar eficazmente el método por firmas, el mismo debe mantener una lista actualizada de las firmas de los ataques conocidos. La lista de firmas de ataques descubiertos sigue creciendo y probablemente nunca deje de crecer. Durante el análisis de mensajes, el WAF debe tomar en consideración toda la lista de firmas en busca de ataques, y esta lista creciente causa que aumente el tiempo de procesamiento y el uso de recursos [6].

El método de detección de anomalías no requiere una lista de firmas, sino que trabaja en dos fases: entrenamiento y detección. En la fase de entrenamiento, este tipo de WAF construye modelos que representan a los mensajes HTTP normales. Así, durante la fase de detección o monitoreo, este tipo de WAF compara los mensajes nuevos con los modelos construidos anteriormente, con el fin de detectar desviaciones significativas, es decir, aquellos mensajes HTTP que son considerados anomalías [6]. La fase de entrenamiento es obligatoria una vez al inicio del uso y después es necesaria si existen cambios en los mensajes normales, por ejemplo, luego de la modificación a una o más aplicaciones web protegidas por el WAF en cuestión. El método por anomalías tiene la ventaja de poder detectar anomalías debidas a nuevos ataques desde el momento que aparezcan, mientras que los métodos por firmas dependen de la actualización de su lista de ataques [6].

Para la detección de anomalías se puede utilizar varias estrategias. Una opción es emplear herramientas estadísticas [6] [4]; otra opción son las herramientas del área de aprendizaje de máquinas (ML - *Machine Learning*) [7] [8]. Una de las áreas de ML son los problemas de clasificación, y la detección de

anomalías puede ser encarada como un problema de este tipo. En estos problemas se busca clasificar las muestras en varios grupos o clases, utilizando una de las herramientas disponibles en ML. Acá también se observan dos fases, una de entrenamiento y otra de detección o clasificación. En este contexto se habla de aprendizaje supervisado si se especifican todas las clases posibles de antemano, usando solamente muestras para el entrenamiento de las que se conocen sus clases; muestras nuevas serán asignadas a la clase a la que más se parezcan. En cambio, se habla de aprendizaje no supervisado cuando no se provee muestras con clases conocidas de antemano y la herramienta debe tratar de encontrar las clases presentes [4]. También se puede dar el caso de que se conozca las clases de solamente algunas de las muestras, o que se tenga únicamente muestras de una clase conocida pero no se tenga muestras de las demás clases; en estos casos se puede hablar de aprendizaje semi-supervisado [9].

Aplicado a un WAF, se puede usar clasificación supervisada, definiendo una clase para los mensajes normales y otra clase (o también varias otras clases) para los mensajes anómalos. Un primer desafío con este abordaje es que se necesita volver a entrenar el clasificador cuando aparece un nuevo tipo de anomalía. Si no se vuelve a entrenarlo con muestras que contengan los nuevos tipos de anomalías, es posible que una anomalía sea clasificada equivocadamente como un mensaje normal en el caso de una anomalía nueva que no se ajusta suficientemente a las clases de anomalías con las cuales el clasificador fue entrenado anteriormente. Un segundo desafío con este abordaje es la necesidad de obtener muestras de todos los tipos de anomalías conocidas para poder realizar un entrenamiento completo.

Estos dos desafíos se trata de superar con la estrategia conocida bajo el nombre de clasificación de una sola clase (OCC - *One-Class Classification*) [10]. Se busca definir una sola clase, la clase conocida, y clasificar las muestras de acuerdo a si pertenecen o no a dicha clase. La fase de entrenamiento utiliza solamente muestras de la clase conocida, con la finalidad de que en la fase de detección las muestras que no se ajusten a la clase conocida sean clasificadas como no perteneciente a la misma. Aplicado a un WAF, la clase conocida esta conformada solamente por los mensajes normales y todos los tipos de anomalías que representan los distintos tipos de ataques no pertenecerán a dicha clase. Para ser consistentes con la terminología del área de seguridad, las anomalías son las muestras positivas, que no pertenecerán a la clase conocida de los mensajes normales (muestras negativas).

El clasificador One-Class SVM es una herramienta ML que ha sido propuesto como una de varias alternativas para afrontar tareas de OCC [11]. Varios investigadores ya han empleado exitosamente este clasificador en problemas de diversas áreas, como por ejemplo clasificación de textos detección de spam, detección de fallas en máquinas, entre otros [12].

Para que un WAF basado en detección de anomalías pueda diferenciar los mensajes HTTP normales de los anómalos, es necesario que existan características de dichos mensajes que posibiliten esa diferenciación. Ejemplos de esos rasgos pueden ser la longitud de la petición, la presencia de ciertos caracteres con significado especial, la distribución de la frecuencia de los caracteres, entre otros [6]. Asumiendo la existencia de esas características distintivas, el éxito del WAF depende de encontrar dichas características y de representarlas en un formato procesable para el mecanismo de detección [4]. En esta parte, el conocimiento experto sobre los mensajes HTTP ayuda a seleccionar las características más útiles para el proceso de detección. Podemos ver un ejemplo de esta selección de características en los trabajos de Kruegel y Vigna [6] [13], donde se utiliza conocimiento sobre la estructura de mensajes para obtener características más específicas y así mejorar los resultados de la detección. Para el clasificador One-Class SVM, esas características de los mensajes HTTP se deben representar con vectores numéricos, llamados también vectores de características o *features*. De esta forma, la eficacia de detección de anomalías del clasificador One-Class SVM depende en gran parte de nuestros procesos de extracción de características, es decir, de los procesos de preprocesamiento que extraen las características distintivas de los mensajes y las representan como vectores numéricos [4].

En este trabajo presentamos OCS-WAF, un sistema para detectar mensajes HTTP anómalos entre las aplicaciones web y sus usuarios con el fin de mitigar los riesgos de ataques contra dichas aplicaciones. Para este fin, OCS-WAF emplea clasificadores One-Class SVM y utiliza conocimiento experto sobre los mensajes para extraer características útiles para la detección de posibles ataques.

II. Extracción de features en el contexto HTTP

II-A. Estructura de mensajes HTTP

El detector OCS-WAF que presentamos en este trabajo analiza mensajes HTTP, limitándose a los mensajes de tipo petición y a la versión 1.1 del protocolo. Las peticiones HTTP pueden estar compuestas por seis partes: un método HTTP, un identificador universal de recursos (URL - *Universal Resource Locator*), un *query string*, la versión del protocolo, varias cabeceras y finalmente el cuerpo de la petición [5]. Esta estructura se puede observar en la Figura 1, que muestra una petición con el método GET y otra con el método POST.

El *query string*, que es opcional, va separado de la URL mediante un signo de interrogación (?) y está compuesto por pares de parámetros y valores. Estos parámetros están separados de sus valores mediante el símbolo de igualdad (=), mientras que los pares se encuentran delimitados por el símbolo *ampersand* (&). De esta forma, el *query string* de una petición puede ser representado como una lista de pares ordenados de parámetros con sus valores.

Las peticiones del método GET no suelen llevar un cuerpo. Las peticiones del método POST en algunos casos lo tienen, y en caso de contar con el mismo, su contenido puede estar estructurado de varias formas posibles. Para esta investigación consideramos únicamente los cuerpos de peticiones POST que constan de pares de parámetros y valores estructurados de la misma forma que el *query string*.

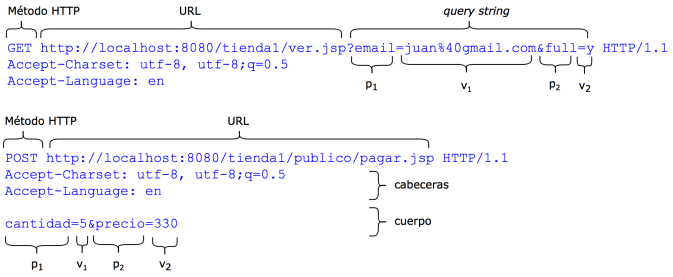

Figura 1: Diagrama de estructura de dos peticiones HTTP, una con método GET y otra con método POST.

II-B. Nuestros procesos de extracción de features

Basado en los trabajos de Kruegel y Vigna [6] [13], nuestros procesos de extracción de *features* también realizan la agrupación de las peticiones HTTP por método y URL. De esta forma, a partir de las peticiones utilizadas para el entrenamiento, se crea un conjunto G de grupos de peticiones, que contiene todos los grupos de peticiones obtenidos mediante la agrupación de los datos de entrenamiento. Con esta agrupación podemos lograr una descripción más precisa de las peticiones normales dentro de cada grupo G_i. Esto posibilita inclusive la identificación de aquellas peticiones que, por ejemplo, son anómalas dentro de su grupo correspondiente G_1, pero que podrían ser consideradas normales en otro grupo G_2.

Con la agrupación realizada, partimos de la base de que las peticiones dentro de un grupo G_i presentan similitud entre ellas. Específicamente, se espera que esas peticiones tengan los mismos parámetros en *query string* y cuerpo, o que haya poca variación. Por ejemplo, en una URL para inicio de sesión, todas las peticiones con el método POST deberían contener los parámetros *username* y *password* en su cuerpo.

De esta forma, en la fase de entrenamiento se construyen listas de todos los parámetros que aparecen en las peticiones HTTP dentro de cada grupo. La lista Q_i contiene de forma ordenada todos los parámetros que aparecen en el *query string* de alguna petición dentro del grupo G_i, excluyendo las duplicaciones. De forma análoga, la lista B_i contiene los parámetros que aparecen en el cuerpo de alguna petición del grupo. Es posible que estas listas queden vacías para un grupo, lo que sucede cuando ninguna petición contiene parámetros. Como ya mencionamos, se espera que los mismos parámetros se repitan en la mayoría de las peticiones, de manera que estas listas de parámetros de un grupo no sean mucho más extensas que la cantidad de parámetros de una sola petición de dicho grupo. Después de construir estas listas, se procesan las peticiones de cada grupo G_i para construir los conjuntos de vectores F_i, en donde cada petición está representada por un vector de *features* $\vec{f_{ij}}$. Por cada petición en G_i, se extraen los valores cuyos parámetros aparezcan en las listas Q_i y B_i. Luego se extraen m *features* de cada uno de esos valores, añadiéndolos de forma ordenada para formar el vector $\vec{f_{ij}}$. Esos m *features* que utilizamos serán explicados en detalle en la siguiente subsección.

Con este análisis de los valores individuales de los parámetros se puede detectar una gran cantidad de ataques que podrían estar presentes en el *query string* o en el cuerpo de las peticiones. Sin embargo, este abordaje no cubre los casos de parámetros que no fueron observados durante la fase entrenamiento, o aquellos ataques ocultos en otras partes de la petición, por ejemplo en las cabeceras de mensajes. Con el fin de mitigar esos riesgos, se extrae también los m *features* de la petición HTTP completa, incluyendo cada una de las seis partes que puede tener dicha petición.

De esta forma, cada petición de G_i queda representada por un vector $\vec{f_{ij}}$, que se encuentra en el espacio $\mathbb{R}^n$. La dimensión n del espacio para cada grupo, representada por n_i, indica la cantidad de componentes de cada $\vec{f_{ij}} \in F_i$ y esta dimensión está dada por la Ecuación 1.

$$n_i = m \times (1 + |Q_i| + |B_i|) \tag{1}$$

En esta ecuación, m es la cantidad de *features* extraídos de cada valor que analizamos; el número 1 representa la petición completa; $|Q_i|$ y $|B_i|$ son las cantidades de parámetros en Q_i y B_i respectivamente. Esta cantidad n_i puede ser distinta en cada grupo G_i, ya que depende de la cantidad de parámetros de las peticiones de esos grupos.

En la fase de detección, para cada petición a analizar se construye un vector de *features*. Se respeta el orden de los parámetros en Q_i y B_i según el grupo G_i correspondiente. Si una petición tiene un parámetro que no se vio en entrenamiento, el valor de este parámetro no es analizado por separado, sino que su contenido solamente se considera dentro del análisis de la petición completa. En cambio, si un parámetro visto durante la fase de entrenamiento no aparece en nuevas peticiones, los vectores llevarán 0 en todos los componentes que correspondan a ese parámetro. De esta forma, los vectores de *features* siempre tendrán la dimensión n_i que corresponde a su grupo.

Dentro de nuestros procesos de extracción de *features* utilizamos 10 números para representar cada valor, resultando en $m = 10$; estos *features* se pueden ver en la Tabla I. Para la extracción de esos *features*, utilizamos tres funciones que retornan uno o más números cada una; específicamente se analiza la distribución de caracteres, la entropía y la cantidad de caracteres.

Features extraídos	Tipo de dato	Rango
Distribución de caracteres - intervalo 0	núm. reales	$[0, 1]$
Distribución de caracteres - intervalo 1	núm. reales	$[0, 1]$
Distribución de caracteres - intervalo 2	núm. reales	$[0, 1]$
Distribución de caracteres - intervalo 3	núm. reales	$[0, 1]$
Distribución de caracteres - intervalo 4	núm. reales	$[0, 1]$
Entropía	núm. reales	$[0, \infty)$
Longitud o cantidad total de caracteres	núm. enteros	$[0, \infty)$
Cantidad de dígitos	núm. enteros	$[0, \infty)$
Cantidad de letras	núm. enteros	$[0, \infty)$
Cantidad de otros caracteres	núm. enteros	$[0, \infty)$

Tabla I: Lista de 10 *features* extraídos por nuestros procesos de extracción de *features* de cada valor analizado.

II-B1. Distribución de caracteres: La distribución de las frecuencias relativas de caracteres puede ser un indicador acerca de la regularidad de la estructura del valor analizado. No se analiza la frecuencia de caracteres individuales, sino que se toma en cuenta la relación entre las frecuencias [6].

Para este fin, primeramente se obtiene las frecuencias relativas de cada carácter distinto que aparece en el valor a analizar, es decir, para cada carácter se cuenta sus apariciones dentro del valor en cuestión y se divide esas cantidades por la longitud del valor, de modo que la suma de todas las frecuencias equivale a 1. Luego, esta lista de frecuencias es ordenada en forma descendente. Se puede esperar que esta distribución de caracteres obtenida tenga una disminución gradual para valores normales. Si un ataque de *buffer overflow* introduciría muchas repeticiones del mismo carácter, la distribución de ese valor tendría una caída pronunciada después de la primera frecuencia en la lista. De forma contraria, si el ataque introduciría un valor largo generado de forma aleatoria, su distribución sería anómala por no poseer variación entre frecuencias.

Los autores Kruegel y Vigna [6] proponen agrupar las frecuencias relativas en intervalos o *bins* de tamaños crecientes. En nuestra implementación, utilizamos cinco intervalos para esta suma de frecuencias; el intervalo 0 contiene la primera frecuencia, el intervalo 1 agrupa la segunda y tercera frecuencia, las siguientes tres frecuencias van al intervalo 2, el intervalo 3 agrupa de la séptima a la décima frecuencia y las demás frecuencias son sumadas en el intervalo 4. Esas cinco sumas obtenidas son cinco componentes del vector $\vec{f_{ij}}$ que representa una petición; posteriormente, el vector será utilizado por el clasificador One-Class SVM para analizar si es una petición normal o anómala.

II-B2. Entropía: La entropía de un valor, en el contexto de la teoría de la información, nos indica la cantidad de información que puede contener dicho valor, relacionando la longitud del valor con la cantidad de símbolos o caracteres distintos presentes en el mismo. La Ecuación 2 muestra la fórmula utilizada para calcular la entropía, propuesta por Claude Shannon hace varias décadas [14].

$$H(x) = -\sum_{i=1}^{n} \left(\frac{c_i}{n} \times \log_2 \frac{c_i}{n} \right) \tag{2}$$

En la ecuación mencionada, el símbolo x es el valor del cual se calcula la entropía, el símbolo n es la cantidad de caracteres distintos presentes en x, mientras que c_i indica la cantidad de apariciones de un mismo carácter en x. La entropía toma valores positivos, donde números cercanos a 0 indican que hay muchas repeticiones de algunos pocos caracteres, mientras que números mayores de entropía resultan de una mayor diversidad de caracteres dentro del valor analizado. Por ejemplo, si comparamos los valores *emp* y *empempempempempempem*, notamos que tienen la misma cantidad de caracteres distintos y la misma entropía, que es $1{,}58$, a pesar de que el segundo tiene mayor longitud. Eso nos muestra que una mayor longitud del valor no es suficiente para obtener mayor entropía, sino que se necesita también mayor variación en su contenido.

En el trabajo [15] se utiliza la entropía como una de las varias métricas empleadas en su sistema de detección de anomalías. Esta medida puede ayudar a detectar, por ejemplo, un valor que contiene un ataque de *buffer overflow* que consta de muchas repeticiones de un solo carácter. Para estos casos la entropía del valor será más baja que en los valores normales observados durante el entrenamiento.

II-B3. Cantidad de caracteres: Kruegel y Vigna [6] utilizan la longitud de valores como un indicador de anomalías. La longitud puede ser entendida también como la cantidad total de caracteres del valor. Nguyen et. al. [15] extraen algunas características de las peticiones HTTP que cuentan un subconjunto de caracteres, por ejemplo, uno de los *features* que utilizan es la cantidad de dígitos presentes en el *path* de la petición, mientras que otro *feature* indica la cantidad total de dígitos presentes en todos los parámetros con sus valores.

Partiendo de los trabajos citados, nuestros procesos de extracción de *features* toman en cuenta las cantidades de cuatro conjuntos distintos de caracteres para cada valor analizado; estos conjuntos son la cantidad total de caracteres, la cantidad de dígitos, de letras, y de otros caracteres que no sean dígitos ni letras. En primer lugar, la cantidad total de caracteres, que es también la longitud del valor, es una información que le brinda la posibilidad al clasificador de detectar valores que sean más largos o cortos que lo normal para cada uno de los parámetro, de la misma forma como lo explican Kruegel y Vigna en sus trabajos citados. Esta información ayuda a detectar, por ejemplo, ataques de *buffer overflow*. Después, las siguientes tres cantidades que se extraen pueden ser útiles para detectar valores que contienen caracteres de conjuntos que son anormales para un parámetro en particular dentro de un grupo de peticiones específico. Por ejemplo, un parámetro que indica el año de nacimiento contendrá valores que consisten de dígitos únicamente. Si en la fase de detección aparece un valor que contiene letras para ese parámetro, nuestro *feature* de cantidad de letras tendrá un número mayor a 0 para ese parámetro de esa petición, pero en la fase de entrenamiento siempre había un 0. Esta información le permite al clasificador detectar la anomalía.

Nuestros procesos de extracción de *features* retornan cuatro números para cada valor analizado, que corresponden a las cantidades de los cuatro conjuntos de caracteres que fueron mencionados anteriormente.

II-C. Composición del vector de features

Durante la fase de entrenamiento, nuestros procesos de extracción de *features* trabajan con una cantidad $|G_i|$ de peticiones recolectadas en cada grupo G_i. El conjunto F_i, que contiene los vectores de *features* del grupo G_i, puede ser expresado también como una matriz numérica M_i, en la cual las filas son los vectores $\vec{f_{ij}}$. Como se puede observar en la Figura 2, esta matriz tendrá una cantidad de filas igual a la cantidad de peticiones utilizadas para el entrenamiento, que es también igual a la cantidad de peticiones en G_i. Además, la cantidad de columnas de M_i será igual a la dimensión n_i de los vectores en F_i.

$$M_i = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n_i} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n_i} \\ \vdots & \vdots & \ddots & \vdots \\ x_{|G_i|,1} & x_{|G_i|,2} & \cdots & x_{|G_i|,n_i} \end{bmatrix}$$

Figura 2: Esquematización de la matriz de *features* M_i del grupo de peticiones G_i que es utilizada para el entrenamiento del clasificador del grupo.

Esta matriz M_i será utilizada para entrenar el clasificador One-Class SVM del grupo. Cabe aclarar que el orden de las filas dentro de la matriz no tiene relevancia para el entrenamiento del clasificador. El orden de las columnas tampoco es importante para el clasificador, siempre y cuando se utilice el mismo orden de *features* durante las dos fases, tanto en la fase de entrenamiento y como también en la fase de detección.

II-D. Escalamiento de los extraídos

Los *features* que extraemos de las peticiones HTTP tienen rangos distintos, como se puede observar en la Tabla I. Un aumento de una unidad en la longitud de la petición completa no puede ser tan importante, pero un aumento de esa misma magnitud en la entropía tiene mayor relevancia. Esta diferencia de impacto que tienen las variaciones de los distintos *features* puede afectar negativamente la capacidad de clasificación de las herramientas de ML. Este potencial problema puede ser mitigado a través de un escalamiento de los *features* extraídos, con el fin de que todos los *features* tengan finalmente rangos similares y, por lo tanto, también la misma importancia [16]. De esta manera, aplicamos un proceso de escalamiento (también denominado normalización por algunos autores) a las matrices M_i antes de pasarlas a los clasificadores One-Class SVM. Mediante este escalamiento se busca que cada *feature* de los datos de entrenamiento, es decir, cada columna de la matriz M_i, tenga un promedio cercano a 0 y una varianza cercana a 1. Estos cálculos son realizados de forma independiente sobre cada uno de los *features*.

Para este proceso, en la fase de entrenamiento se calcula el promedio y la desviación estándar de cada columna de la matriz M_i y luego se aplica el cálculo de la Ecuación 3 a cada uno de los valores de dicha matriz.

$$x_{\text{nuevo}} \;=\; \frac{x_{\text{actual}} - \mu}{\sigma} \tag{3}$$

Como se puede observar en esta ecuación, a cada uno de los valores x de la matriz M_i se le resta μ, que es el promedio de los valores de la columna correspondiente; ese resultado de la resta es dividido por σ, que es la desviación estándar correspondiente a los valores de la columna.

Los valores μ y σ, que fueron calculados para cada columna durante el entrenamiento, son almacenados para utilizarlos posteriormente en la fase de detección; en esa fase se aplica el mismo cálculo de escalamiento a los componentes de los vectores de *features* que representan a las nuevas peticiones, utilizando los valores μ y σ calculados.

III. Clasificación con One-Class SVM

El One-Class SVM es un clasificador de aprendizaje semi-supervisado. En su fase de entrenamiento recibe solamente muestras de la clase conocida y traza un hiperplano que separa las muestras del origen de coordenadas. Posteriormente, en la fase de detección, el clasificador analiza de qué lado del hiperplano se encuentran las nuevas muestras; si están situadas en el lado opuesto al origen, serán consideradas como pertenecientes a la clase conocida, pero en el caso contrario, no pertenecerán a dicha clase [17]. En nuestro contexto de detección de anomalías HTTP, la clase conocida serán las peticiones normales y todas las anomalías no pertenecerán a dicha clase.

III-A. Fase de entrenamiento

En esta fase, nuestro WAF representa las peticiones HTTP por medio de vectores numéricos de *features* $\vec{f_{ij}}$ y entrena un One-Class SVM por cada grupo G_i, utilizando las matrices M_i para dicho entrenamiento.

El One-Class SVM busca un hiperplano que separa las muestras (en nuestro caso, los vectores de *features* $\vec{f_{ij}}$) del origen. Este hiperplano puede ser expresado según la Ecuación 4 [18], donde $\vec{w_i}$ es el vector perpendicular al hiperplano, $\vec{x}$ es un vector variable y ρ_i indica la distancia del hiperplano al origen para el clasificador del grupo G_i.

$$\vec{w_i} \cdot \vec{x} \;-\; \rho_i \;=\; 0 \tag{4}$$

Para obtener el hiperplano óptimo, se busca resolver el problema de optimización de la Ecuación 5, sujeto a las restricciones de la Ecuación 6 [18].

$$\min_{\vec{w_i}\ ,\ \rho_i\ ,\ \xi_i} \; \frac{1}{2}\|\vec{w_i}\|^2 \;-\; \rho_i \;+\; \frac{1}{\nu_i |G_i|} \sum_{j=1}^{|G_i|} \xi_{ij} \tag{5}$$

$$\vec{w_i} \cdot \vec{f_{ij}} \geqslant \rho_i - \xi_{ij} \;, \quad \xi_{ij} \geqslant 0 \;, \quad \forall j = 1, 2, \ldots, |G_i| \tag{6}$$

El parámetro fijo ν_i regula la fracción de muestras situadas al mismo lado del hiperplano que el origen (serán falsos positivos) para el grupo G_i; esto provee robustez frente a la presencia de ruido en los datos de entrenamiento. La variable ξ_i es una lista de penalizaciones incurridas por muestras situadas al mismo lado del hiperplano que el origen. El problema de minimización busca maximizar la distancia ρ_i del hiperplano al origen, mientras que también busca minimizar las penalizaciones por muestras de entrenamiento situadas al mismo lado del hiperplano que el origen. El clasificador puede balancear entre el aumento de ρ_i (alejando el hiperplano del origen) y la reducción de las penalizaciones en ξ_i (reduciendo falsos positivos) para obtener el hiperplano óptimo [18].

Dependiendo del conjunto de muestras utilizadas para el entrenamiento, el clasificador One-Class SVM no siempre podrá encontrar un hiperplano adecuado para separar las muestras del origen. Una solución a esta situación es mapear las muestras a un espacio vectorial de dimensiones mayores para encontrar un hiperplano separador en ese nuevo espacio

[18]. Esta transformación puede ser expresada como funciones $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$, que llevan las muestras del espacio original $\mathbb{R}^n$ a otro espacio $\mathbb{R}^m$, con $n \leq m$. El hiperplano trazado por el clasificador en el espacio $\mathbb{R}^m$ puede quedar representado por una superficie no lineal en el espacio original $\mathbb{R}^n$.

Esta transformación a espacios superiores trae consigo la dificultad de realizar las multiplicaciones de vectores en dichos espacios, que puede provocar que el clasificador consuma demasiados recursos o necesite mucho tiempo de computación, haciendo que el uso del clasificador en ambientes reales ya no sea posible [16]. Para superar este desafío, se puede emplear unas funciones denominadas *kernels*. El resultado producido por estas funciones equivale al producto escalar de dos vectores en el nuevo espacio $\mathbb{R}^m$, pero con la ventaja de que no se realiza realmente esa multiplicación costosa [16].

Nosotros utilizamos el *Radial Basis Function* (RBF) *kernel* en nuestra implementación; su formulación se puede ver en la Ecuación 7 [18], donde el parámetro γ_i determina la influencia de las muestras de entrenamiento del grupo G_i para el posicionamiento del hiperplano.

$$K_i(\vec{x_1}, \vec{x_2}) \;=\; \phi(\vec{x_1}) \cdot \phi(\vec{x_2}) \;=\; \exp(-\gamma_i \|\vec{x_1} - \vec{x_2}\|^2) \quad (7)$$

Para aprovechar la facilidad de cálculo que brindan los *kernels*, es conveniente que el One-Class SVM no resuelva directamente el problema de optimización de la Ecuación 5, sino que resuelva la formulación dual del problema según se muestra en la Ecuación 8, sujeto a las restricciones que muestra la Ecuación 9 [9].

$$\min_{a_i} \; \frac{1}{2} \; a_i \; S_i \; a_i^T \quad (8)$$

$$\sum_{j=1}^{|G_i|} a_{ij} \;=\; 1 \;, \quad 0 \leqslant a_{ij} \leqslant \frac{1}{\nu_i |G_i|} \; \forall j = 1, 2, \ldots, |G_i| \quad (9)$$

S_i es una matriz simétrica que representa los productos escalares de todas las muestras de entrenamiento del grupo G_i multiplicadas entre ellas en el espacio de dimensiones mayores $\mathbb{R}^m$; esta matriz puede ser reemplazada por los resultados del *kernel*. El vector a_i contiene un coeficiente variable para cada una de las muestras, y la formulación dual de la optimización busca encontrar estos coeficientes [9].

Después de resolver la minimización de la formulación dual, el hiperplano del grupo G_i puede ser expresado en términos del *kernel* K_i utilizado y del vector a_i obtenido, como lo muestra la Ecuación 10 [9]. Acá, $\phi(\vec{x})$ es el vector variable transformado al espacio $\mathbb{R}^m$.

$$\vec{w_i} \cdot \phi(\vec{x}) \;-\; \rho_i \;=\; \sum_{j=1}^{|G_i|} \Big(a_{ij} \, K_i(\vec{f_{ij}}, \vec{x}) \Big) \;-\; \rho_i \;=\; 0 \quad (10)$$

III-B. Fase de detección

Después de la fase de entrenamiento, el One-Class SVM está preparado para clasificar nuevas muestras, analizando de que lado del hiperplano se encuentran las mismas. Si están situadas en el lado opuesto al origen, serán consideradas como pertenecientes a la clase conocida. En caso contrario, no pertenecerán a dicha clase [17]. En nuestro contexto de detección de anomalías, el WAF representa cada nueva petición HTTP con un vector de *features* con dimensiones correspondientes al grupo G_i según método y URL de la petición, y luego analiza la posición de ese vector con respecto al hiperplano del grupo. Esto queda expresado como la función de decisión $g_i(\vec{x})$ de la Ecuación 11 [19].

$$g_i(\vec{x}) \;=\; \begin{cases} \vec{w_i} \cdot \phi(\vec{x}) - \rho_i \;\geqslant\; 0 & +1 \\ \text{caso contrario} & -1 \end{cases} \quad (11)$$

Empleando las optimizaciones que nos brindan los *kernels*, podemos sustituir la multiplicación de vectores en esta función. Así, la función de decisión que expresada según la Ecuación 12 [17].

$$g_i(\vec{x}) \;=\; \begin{cases} \sum_{j=1}^{|G_i|} \Big(a_{ij} \, K_i(\vec{f_{ij}}, \vec{x}) \Big) - \rho_i \;\geqslant\; 0 & +1 \\ \text{caso contrario} & -1 \end{cases} \quad (12)$$

De esta forma, se obtendrá $g_i(\vec{x}) = 1$ para las muestras que se encuentran separadas del origen por el hiperplano, indicando que pertenecen a la clase conocida (serán identificadas como peticiones normales). En el caso contrario, se obtendrá $g_i(\vec{x}) = -1$ para las muestras que se encuentren en el mismo lado que el origen, indicando que estas no pertenecen a dicha clase (serán identificadas como anomalías o ataques).

IV. Implementación de OCS-WAF

IV-A. Arquitectura de nuestra implementación

El detector OCS-WAF, implementado en el marco de este trabajo, consiste en un *proxy* HTTP que contiene herramientas para el análisis de las peticiones que pasan por el mismo. Este WAF es colocado frente a las aplicaciones web a ser protegidas, de forma que todo el tráfico entrante y saliente de dichas aplicaciones pase por este dispositivo de detección. Se puede observar dos fases diferentes en OCS-WAF, una de entrenamiento y otra de detección. La fase de entrenamiento es un proceso en el cual se utilizan peticiones HTTP recolectadas que representan el tráfico normal de las aplicaciones web para entrenar los One-Class SVM (un clasificador por grupo de peticiones). En la fase de detección, OCS-WAF analiza nuevas peticiones entrantes mediante los clasificadores entrenados con el fin de determinar si dichas peticiones son normales (muestras de la clase conocida) o anómalas (muestras positivas o ataques). Ambas fases incluyen un paso de preprocesamiento, que consiste en una extracción de *features* para representar las peticiones con vectores numéricos.

OCS-WAF es una implementación sencilla hecha en el lenguaje de programación *Python* en su versión 3.5. La base

está compuesta por un *proxy* HTTP creado por Philippe Lagadec [20]. Fueron realizadas algunas modificaciones a esta base de forma que funcione con nuestra versión de *Python*, ya que originalmente fue escrito para *Python* 2. Esta base realiza todas las tareas de recibir las peticiones HTTP y enviarlas a las aplicaciones web destino, como también realiza el proceso inverso para hacer llegar las respuestas de las aplicaciones de vuelta al origen de las peticiones. Además, la base provee una simple interfaz de funciones que fueron extendidas para realizar el análisis de las peticiones. OCS-WAF solamente analiza las peticiones entrantes, pero nuestra implementación puede ser extendida para incluir el análisis de las respuestas que son retornadas por las aplicaciones protegidas. OCS-WAF también utiliza la librería *scikit-learn* [21], que es una librería con herramientas del área de ML para *Python*. De esta librería se utilizaron varias herramientas, como por ejemplo, el *BaseEstimator* para implementar las funciones de extracción de *features*, el *StandardScaler* para el escalamiento de *features*, el *OneClassSVM* para el proceso de clasificación y finalmente se emplea el *FeatureUnion* y el *Pipeline* para coordinar el flujo de datos entre todos estos componentes.

IV-B. Fase de entrenamiento

Esta fase sirve para preparar el OCS-WAF para la detección de peticiones HTTP anómalas. La Figura 3 muestra la arquitectura de OCS-WAF en esta fase de entrenamiento. Se utiliza peticiones recolectadas que representan el tráfico normal de las aplicaciones web, las cuales pasan por tres pasos intermedios durante esta fase; primeramente se agrupan las peticiones según su método HTTP y URL, luego se realiza el preprocesamiento de las peticiones y finalmente se entrena los clasificadores dentro de cada grupo.

Como se puede observar en la figura mencionada, para esta fase se necesitan peticiones normales (la clase conocida) que representan el tráfico normal que se espera recibir durante la fase de detección. Se podría usar nuestra implementación para esta recolección de muestras, deshabilitando los procesos de análisis y almacenando las peticiones que pasen por el detector.

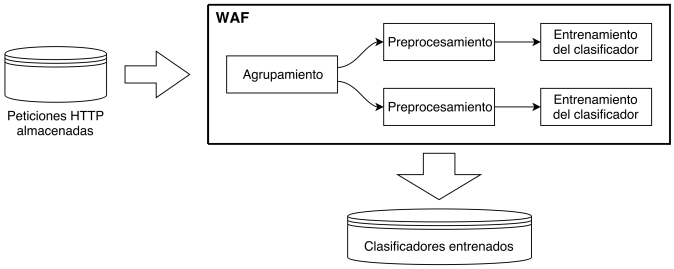

Figura 3: Arquitectura de OCS-WAF en la fase de entrenamiento.

IV-B1. Paso de agrupamiento: Agrupamos las peticiones HTTP por método y URL para aprovechar las semejanzas que presentan las peticiones dentro de un mismo grupo. Partimos de la premisa de que peticiones que van dirigidas a una misma URL y con el mismo método HTTP presentan más similitudes entre ellas que con peticiones que tienen otro método o URL. Agrupando las peticiones disponibles por método y URL se puede entrenar clasificadores independientes sobre cada uno de estos grupos. De esta forma se puede obtener modelos de anomalías más precisos dentro de cada grupo, con el fin de lograr mejores resultados en la fase de detección. Podemos denominar G al conjunto de los grupos de peticiones que se obtienen por este paso de agrupamiento y cada grupo es identificado como G_i.

IV-B2. Paso de preprocesamiento: En el paso de preprocesamiento, nuestros procesos de extracción de *features* son aplicados a las peticiones HTTP para representarlas con vectores numéricos de *features*. Este paso se realiza de forma independiente para cada grupo G_i. Debido a eso, en la Figura 3 podemos ver múltiples instancias de preprocesamiento, que corresponden a los distintos grupos.

Primeramente, OCS-WAF construye las listas Q_i y B_i, que contienen todos los parámetros que aparecen en el *query string* y cuerpo de alguna petición dentro del grupo G_i. Las duplicaciones son excluidas, y las listas son ordenadas de forma alfabética. Después, nuestra implementación procesa las peticiones de cada grupo G_i para construir los conjuntos de vectores F_i, en donde cada petición está representada por un vector de *features* $\vec{f_{ij}}$. Así, por cada petición en G_i, OCS-WAF extrae los $m = 10$ *features* de la petición completa, incluyendo cada una de las seis partes que puede tener dicha petición. Luego se extraen los valores cuyos parámetros aparezcan en las listas Q_i y B_i, y se generan $m = 10$ *features* de cada uno de esos valores.

Los $m = 10$ *features* extraídos analizan distintas características, que son la distribución de caracteres, la entropía y la cantidad de caracteres de cada valor, que ya fueron explicados en detalle en la Sección II. Todos los números retornados por estas funciones son concatenados de forma ordenada para formar el vector $\vec{f_{ij}}$ de cada petición. La dimensión de estos vectores está definida por la Ecuación 1. Cabe resaltar que cada grupo G_i puede tener una dimensión distinta para sus vectores $\vec{f_{ij}}$, ya que depende de la cantidad de parámetros de las peticiones del grupo. Con estos vectores de *features* se construye la matriz M_i de cada grupo, donde los vectores conforman las filas de la matriz. Luego de obtener la matriz M_i del grupo, se procede a escalar los *features*, que son las columnas de la matriz. Se busca que cada *feature* tenga promedio cercano a 0 y varianza cercana a 1.

Aclaramos que OCS-WAF puede ser extendido con mucha facilidad en esta parte de extracción de *features*, agregando o quitando funciones que analizan los valores.

IV-B3. Paso de entrenamiento del clasificador: Después de realizar la extracción de *features* y el escalamiento de los mismos, OCS-WAF utiliza las matrices M_i obtenidas para entrenar un clasificador One-Class SVM para cada grupo G_i. Para el proceso de entrenamiento, el One-Class SVM recibe la matriz M_i y también los valores para los parámetros ν_i y γ_i elegidos para ese grupo. Utilizamos el *kernel* RBF para todos los clasificadores. Para finalizar esta fase

de entrenamiento, OCS-WAF almacena toda la información necesaria para realizar la detección de anomalías en la fase de detección. Primeramente se almacenan datos generados por los procesos de extracción de *features*, que incluye los parámetros extraídos y su orden dentro de los vectores, luego se persiste los promedios y desviaciones calculados para el escalamiento de cada *features*, y por último se guarda también el clasificador entrenado. Nuestra implementación almacenado toda esta información en un archivo binario en disco, para ser utilizado posteriormente en la fase de detección.

La selección de los valores para los parámetros ν_i y γ_i presenta un gran desafío [10]. Una estrategia para encontrar valores adecuados para los dos parámetros en cuestión puede ser el entrenamiento de un clasificador con solamente un subconjunto de los datos disponibles, utilizando los datos restantes para una validación posterior al entrenamiento. Este proceso puede ser repetido para varios valores distintos de ν_i y γ_i, seleccionando finalmente los valores que resulten en la mejor clasificación de los datos de entrenamiento. Utilizamos esta estrategia en las pruebas que realizamos con OCS-WAF, las cuales presentaremos en el siguiente capítulo. Sin embargo, nuestra implementación no cuenta todavía con un método automático para la selección de los valores para ν_i y γ_i.

IV-C. Fase de detección

En esta fase, OCS-WAF utiliza los procesos de extracción y los clasificadores entrenados para analizar las peticiones HTTP entrantes con el fin de determinar si dichas peticiones son normales o anómalas. Según el resultado de la detección, el WAF realiza distintas acciones configurables para cada caso. En la Figura 4 se puede observar la arquitectura de OCS-WAF en esta fase de detección. Se puede notar cuatro pasos intermedios para esta fase, que son el enrutamiento de las peticiones a su grupo G_i correspondiente, el preprocesamiento y la clasificación de las mismas, y finalmente las acciones que son realizadas en respuesta al resultado de la clasificación de cada petición analizada.

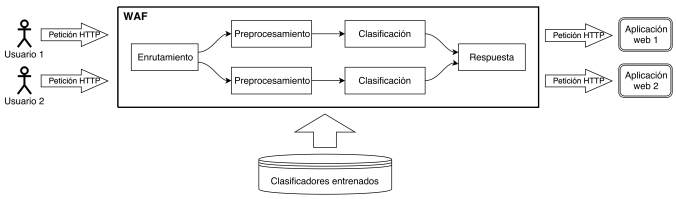

Figura 4: Arquitectura de OCS-WAF en la fase de detección.

IV-C1. Paso de enrutamiento: Como se puede observar en la Figura 4, OCS-WAF puede proteger múltiples aplicaciones web y procesar peticiones de varios usuarios. Los grupos G_i, que fueron formados durante la fase de entrenamiento, pueden ser identificados a través del método HTTP y la URL de sus peticiones. Para proveer una detección más rápida, OCS-WAF en su proceso de inicialización ya carga en memoria toda la información almacenada del entrenamiento, para de esta forma evitar las costosas lecturas de disco durante el proceso de detección. De esta forma, este primer paso de enrutamiento determina el grupo G_i correspondiente a cada nueva petición que es recibida por el WAF.

En caso de que no se encuentre el grupo para una petición, OCS-WAF puede simplemente reenviar dicha petición a su destino por no tener clasificadores para analizarla, o de lo contrario, puede bloquear dicha petición por no haber visto peticiones para ese grupo durante el entrenamiento. Esta configuración queda a cargo de los administradores responsables por el detector.

IV-C2. Paso de preprocesamiento: Una vez determinado el grupo G_i correspondiente a una nueva petición, OCS-WAF procede a aplicar los procesos de extracción de *features* a la misma.

Primeramente, se extraen los $m = 10$ *features* de la petición completa. Luego, utilizando las listas Q_i y B_i del grupo, los valores de los parámetros de la petición son extraídos y se generan los $m = 10$ *features* de cada uno. Atendiendo el orden de los *features*, que fue establecido durante el entrenamiento, OCS-WAF construye el vector $\vec{x}$ de la nueva petición. Este nuevo vector tendrá la dimensión correspondiente al grupo. Luego se aplica el proceso de escalamiento al vector, usando el promedio y la desviación estándar que fueron calculados para el grupo G_i correspondiente durante el entrenamiento.

IV-C3. Paso de clasificación: En este paso, OCS-WAF emplea el clasificador One-Class SVM entrenado del grupo correspondiente para determinar si la nueva petición será considerada normal o anómala. El paso de clasificación consiste en aplicar la función de decisión $g_i(\vec{x})$ al vector de *features* $\vec{x}$ que representa a la nueva petición en cuestión. Si la clasificación obtiene $g_i(\vec{x}) = 1$, entonces la representación de esta nueva petición en el espacio de dimensiones mayores se encuentra separada del origen por el hiperplano, indicando que esta petición es normal (una muestra negativa). En cambio, si se obtiene $g_i(\vec{x}) = -1$, entonces la nueva petición se encuentra del mismo lado que el origen, indicando que se trata de una anomalía (una muestra positiva o ataque).

IV-C4. Paso de respuesta: Después de clasificar la nueva petición como normal o anómala, OCS-WAF procede a realizar las acciones de respuesta que fueron configuradas para el resultado de clasificación correspondiente. Si la petición en cuestión es considerada normal, la misma es reenviada a la aplicación web destino. En cambio, si la petición es considerada anómala, el resultado de la clasificación y la petición quedan registrados en un *log*. Opcionalmente, OCS-WAF puede ser configurado para bloquear las peticiones anómalas, evitando de esta forma que las mismas lleguen a las aplicaciones destino. En cambio, si el bloqueo se encuentra deshabilitado, la petición anómala es reenviada a la aplicación destino, como si fuera una petición normal. La configuración del bloqueo de peticiones anómalas queda a cargo de los administradores responsables, ya que los distintos ambientes de implementación podrían tener necesidades diferentes. Además, nuestra implementación puede ser extendida con otras acciones a realizar en respuesta a la detección de peticiones anómalas. Por ejemplo, se podría agregar el envío de notificaciones o alarmas en caso de anomalías.

V. Pruebas y resultados

V-A. Conjuntos de datos de prueba

Para evaluaciones cuantitativas de la eficacia de detección del WAF que hemos implementado necesitamos conjuntos de datos. Obtener buenos datos para las pruebas es fundamental para obtener resultados que después se comprueben en el mundo real [4]. Utilizamos los conjuntos de datos CSIC 2010 [22] y CSIC TORPEDA 2012 [23], que contienen peticiones HTTP hechas a una aplicación sencilla de comercio electrónico, simulando distintos escenarios como, por ejemplo, registro de clientes y compra de productos. Varias investigaciones relacionadas ya utilizaron estos conjuntos, como por ejemplo Parhizkar y Abadi [18] y Torrano-Giménez [4], y así podemos comparar resultados con dichos trabajos.

Considerando que nuestro WAF entrena un clasificador por cada combinación de URL y método HTTP, nosotros agrupamos todas las peticiones de los conjuntos de datos, limitándonos a aquellos grupos que tienen más de 100 peticiones normales y también más de 100 peticiones anómalas. Así obtuvimos 18 grupos, alcanzando un total de $40\,130$ peticiones normales y $42\,444$ anómalas.

V-B. Análisis de la eficacia de detección

Realizamos pruebas para analizar la eficacia de detección de OCS-WAF, utilizando los conjuntos de datos presentados. Para la cuantificación de las pruebas, utilizamos peticiones normales (N, o muestras negativas) y anomalías o ataques (P, o muestras positivas). Las peticiones normales pueden ser detectadas como tales, resultando en negativos correctos (TN), o pueden ser detectadas como anomalías, resultando en falsos positivos (FP). De la misma forma, las peticiones anómalas pueden ser detectadas como tales, resultando en positivos correctos (TP), o pueden ser detectadas erróneamente como normales, resultando en falsos negativos (FN).

Para el análisis de nuestros resultados utilizamos varias métricas que son fórmulas agregadas de estas cantidades vistas. En primer lugar, el *True Positives Rate* (TPR) indica la fracción de peticiones anómalas (muestras positivas) que fue detectada correctamente; tiene un rango $[0; 1]$. Este valor debería ser cercano a 1 para que el WAF detecte la mayor cantidad posible de ataques antes de que lleguen a las aplicaciones protegidas. En segundo lugar, el *False Positives Rate* (FPR) indica la fracción de peticiones normales (muestras negativas) que fue detectada equivocadamente como anomalías; tiene un rango $[0; 1]$. Este valor debería ser cercano a 0 para que el WAF no afecte negativamente la experiencia de usuarios legítimos de las aplicaciones protegidas al bloquear peticiones normales. Finalmente, el F_1-*score* busca incorporar las dos métricas anteriores en un solo indicador; tiene también un rango $[0; 1]$. Este valor debería ser también cercano a 1 para cumplir los dos objetivos mencionados acerca del TPR (detectar todos los ataques) y FPR (no afectar usuarios legítimos). Estas tres métricas pueden ser expresadas también como muestra la Ecuación 13.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}}\ , \quad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}}\ , \quad \mathrm{F}_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{13}$$

Realizando distintas pruebas, nuestros resultados confirman que el escalamiento de los *features* y el análisis de valores de los parámetros mejora la capacidad de detección del OCS-WAF. También utilizamos 200, 500, $1\,000$ y $1\,500$ peticiones para el entrenamiento (que equivale a $10\,\%$, $25\,\%$, $50\,\%$ y $75\,\%$ de los datos disponibles en la mayoría de los grupos), obteniendo los mejores resultados para la mayor cantidad de peticiones utilizadas, aunque las diferencias comparado a las demás cantidades son pequeñas y se podría usar también menos peticiones sin una degradación significativa de la eficacia de detección. Además, investigando la influencia de anomalías en los datos de entrenamiento, los resultados muestran que OCS-WAF con sus clasificadores One-Class SVM presenta cierta robustez frente a la presencia de algunas pocas anomalías en los datos de entrenamiento. En la Tabla II se puede observar los resultados de detección de OCS-WAF, que fueron obtenidos usando el escalamiento de los *features* con $1\,500$ peticiones para el entrenamiento, incluyendo el análisis de valores de los parámetros. Como promedio de los 18 grupos de peticiones logramos obtener un TPR de $0{,}93$, un FPR de $0{,}03$ y un F_1-*score* de $0{,}95$.

Grupo	TPR	FPR	F_1-*score*
c00	$0{,}71 \pm 0{,}01$	$0{,}05 \pm 0{,}00$	$0{,}80 \pm 0{,}00$
c01	$0{,}72 \pm 0{,}01$	$0{,}05 \pm 0{,}01$	$0{,}80 \pm 0{,}00$
c02	$1{,}00 \pm 0{,}00$	$0{,}03 \pm 0{,}01$	$0{,}98 \pm 0{,}01$
c03	$1{,}00 \pm 0{,}00$	$0{,}03 \pm 0{,}00$	$0{,}98 \pm 0{,}00$
c04	$0{,}91 \pm 0{,}01$	$0{,}01 \pm 0{,}00$	$0{,}95 \pm 0{,}01$
c05	$0{,}92 \pm 0{,}01$	$0{,}01 \pm 0{,}00$	$0{,}95 \pm 0{,}00$
c06	$0{,}99 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
c07	$0{,}99 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
c08	$1{,}00 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
c09	$1{,}00 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
c10	$1{,}00 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
c11	$1{,}00 \pm 0{,}00$	$0{,}01 \pm 0{,}00$	$0{,}99 \pm 0{,}00$
c12	$0{,}74 \pm 0{,}00$	$0{,}05 \pm 0{,}01$	$0{,}81 \pm 0{,}01$
c13	$0{,}74 \pm 0{,}00$	$0{,}05 \pm 0{,}01$	$0{,}81 \pm 0{,}01$
c14	$1{,}00 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
c15	$1{,}00 \pm 0{,}00$	$0{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$
t00	$0{,}99 \pm 0{,}01$	$0{,}06 \pm 0{,}04$	$0{,}98 \pm 0{,}00$
t01	$1{,}00 \pm 0{,}00$	$0{,}09 \pm 0{,}06$	$0{,}99 \pm 0{,}01$
	$0{,}93 \pm 0{,}11$	$0{,}03 \pm 0{,}03$	$0{,}95 \pm 0{,}08$

Tabla II: Resultados de detección de OCS-WAF.

V-C. Análisis del tiempo de respuesta de las aplicaciones protegidas

Esta prueba que realizamos con OCS-WAF apunta a cuantificar en qué medida se ve afectado el tiempo de respuesta de las aplicaciones web que nuestra implementación debe proteger. Creamos una simple aplicación capaz de recibir peticiones HTTP y enviar las respuestas correspondientes, con el fin de simular aplicaciones web protegidas por OCS-WAF. Después, medimos el intervalo de tiempo entre el envío de una petición a esa aplicación y la recepción de la respuesta.

Realizamos esta medición con tres configuraciones distintas. En la primera configuración se enviaron las peticiones directamente a la aplicación destino sin pasar por OCS-WAF, para obtener así el tiempo de respuesta normal. En la segunda configuración el tráfico fue enviado a través del WAF pero sin que este realice las tareas de detección, para de esta manera medir el retraso incurrido por agregar un componente al camino de los paquetes. En la última configuración las peticiones pasaron por un WAF previamente entrenado, con el fin de medir el retraso ocasionado al utilizar las funciones de detección.

La configuración sin OCS-WAF obtuvo el menor tiempo de respuesta, con un promedio de $4{,}8$ milisegundos. En la segunda configuración, que utiliza OCS-WAF sin detección, se obtuvo un retardo adicional de $1{,}6$ milisegundos en tiempo de respuesta comparado al primer caso. La tercera configuración, que incluye la detección, obtuvo retardos adicionales de $5{,}3$ y $3{,}7$ milisegundos comparado con las dos primeras configuraciones respectivamente. Esto indica el orden de magnitud del retraso que OCS-WAF podría agregar a las peticiones. Se espera que esto sea un límite superior para un WAF optimizado (incluso utilizando otro lenguaje). Además, considerando que la latencia promedio de paquetes de red en Internet está entre 200 y 300 milisegundos [24] y será un poco mayor a eso para mensajes HTTP, se espera que estos pocos milisegundos adicionales agregados por OCS-WAF no afectarán de forma notable el tiempo de respuesta de las aplicaciones.

V-D. Análisis del tiempo de entrenamiento

En esta última prueba realizada se analizó la relación que existe entre el tiempo de entrenamiento de OCS-WAF (que incluye el tiempo consumido por los procesos de extracción de *features* y los clasificadores One-Class SVM empleados) y la cantidad de peticiones HTTP utilizadas para dicho entrenamiento. La duración estimada del entrenamiento es una información importante para los administradores responsables por OCS-WAF, ya que estas personas necesitan saber si el entrenamiento durará minutos, horas o días, a fin de poder planificar los momentos adecuados para este proceso.

Nuestros resultados indican que el tiempo de entrenamiento máximo está en el orden de algunos milisegundos por petición, específicamente el máximo está en 7 milisegundos. Esto significa que el entrenamiento de OCS-WAF puede durar 704 segundos (cerca de 12 minutos) para una cantidad de $100\,000$ peticiones. En nuestra opinión, este tiempo es muy manejable, permitiendo que el WAF pueda ser entrenado con nuevos datos en cuestión de minutos en los momentos que el administrador del mismo lo considere necesario.

V-E. Comparación con trabajos relacionados

Comparamos nuestro implementación y los resultados que obtuvimos con algunos trabajos de autores relacionados al área de IDS con detección de anomalías. Nos enfocamos en trabajos que utilizan los mismos conjuntos de datos que nosotros empleamos y en la Tabla III se muestra un resumen de esta comparación.

Detector	TPR	FPR	F_1-*score*
OCS-WAF (el presente trabajo)	0,93	0,03	0,95
ModSecurity (firmas de ataques) Giménez [1]	0,56	0,00	0,71
HTTP-WS-AD (métodos estadísticos) Giménez [1]	0,99	0,02	0,99
Conjunto de árboles de decisión Torrano-Giménez [4]	0,95	0,05	-
OC-WAD (usa One-Class SVM) Parhizkar y Abadi [18]	0,96	0,03	-

Tabla III: Comparación de resultados de OCS-WAF con otros trabajos que utilizan también el conjunto de datos CSIC 2010.

VI. Conclusiones

VI-A. Resumen de la investigación

En la Sección I introducimos el tema de este trabajo. Describimos el área de estudio que rodea nuestra investigación, que abarca el área de IDS con detección de anomalías, características de los mensajes HTTP y también la herramienta One-Class SVM del área de ML. Luego mostramos la problemática que nos llevó a realizar este trabajo, argumentando que vulnerabilidades presentes en muchas aplicaciones web presentan un gran riesgo y se necesita mecanismos para protegerlos. Explicamos que este trabajo se centra en la detección de mensajes HTTP anómalos mediante un WAF basado en detección de anomalías, utilizando clasificadores One-Class SVM, con el fin de mitigar los riesgos existentes de ataques contra aplicaciones web a través de la Internet.

En la Sección II presentamos nuestros procesos de extracción de *features*. Estos procesos se basan de gran manera en los aportes de los trabajos de Kruegel y Vigna, especialmente en la manera como aprovechan las características de mensajes HTTP para proponer sus modelos de anomalías. Luego explicamos los procesos que diseñamos, incluyendo el análisis de valores de parámetros que realizamos y cada uno de los *features* que extraemos de esos valores para representar las características de los mensajes como vectores numéricos. Estos vectores pueden tener longitudes distintas según el grupo de método HTTP y URL al que pertenecen, ya que esto depende de la cantidad de parámetros de las peticiones. Las características analizadas de las peticiones son la distribución de caracteres, la entropía y la cantidad de caracteres. Se complementa esta extracción de *features* con un proceso de escalamiento para reducir el impacto de los distintos rangos de números que pueden tener los diferentes *features*.

En la Sección III describimos en detalle el clasificador One-Class SVM, que constituye la parte central de nuestro WAF. Una de las ventajas de utilizar este clasificador es que no necesitamos peticiones anómalas para el entrenamiento, reduciendo así el impacto que tiene la aparición de nuevos tipos de ataques; solamente se vuelve a realizar el entrenamiento cuando haya cambios en las aplicaciones protegidas. Mostramos que se utiliza vectores de *features* que representan a las peticiones de entrenamiento para trazar un hiperplano que separa este grupo de entrenamiento del origen. Posteriormente,

en la fase de detección, se determina a qué lado del hiperplano pertenecen los vectores de las nuevas peticiones, para así determinar si corresponden a peticiones normales o anómalas. Observamos que la selección de valores de los parámetros ν_i y γ_i del clasificador se realizó de forma experimental para los conjuntos de datos utilizados y creemos que esto no variará mucho para otros conjuntos de datos diferentes.

La Sección IV detalla la implementación y funcionamiento de OCS-WAF que fue construido en el marco de este trabajo. Describimos los detalles de la implementación, incluyendo los pasos intermedios de cada una de las dos fases, que son las fases de entrenamiento y detección. Se explicaron las particularidades de nuestra implementación, detallando también los componentes externos y librerías que utilizamos. El código fuente de la implementación está disponible en un repositorio público bajo la dirección `https://github.com/nico-ralf-ii-fpuna/tfg`.

En la Sección V presenta las pruebas realizadas con nuestra implementación y expone los resultados obtenidos en cada una de ellas. Utilizamos los conjuntos de datos CSIC 2010 [22] y CSIC TORPEDA 2012 [23], que contienen peticiones HTTP hechas a una aplicación de comercio electrónico. Las pruebas experimentales analizaron tres características de OCS-WAF: la eficacia de detección, el impacto en el tiempo de respuesta de las aplicaciones protegidas y el tiempo de entrenamiento.

Analizando la primera de estas característica, la eficacia de detección de OCS-WAF, logramos obtener un TPR de $0{,}93$, un FPR de $0{,}03$ y un F_1-*score* de $0{,}95$. Estos son resultados promedios para los 18 grupos de peticiones que utilizamos de los conjuntos de datos. Empleamos el escalamiento de *features* e incluimos el análisis de valores de parámetros en las peticiones, ya que utilizando estas dos estrategias obtuvimos los mejores resultados. Además, realizamos el entrenamiento con 1500 peticiones en cada grupo (cerca del 75 % de las muestras normales), atendiendo a que no haya anomalías en los datos de entrenamiento. Notamos que el TPR promedio alcanzado de $0{,}93$ es algo bajo; inspeccionando los grupos individuales descubrimos que la mayoría tiene un TPR mayor o igual a $0{,}99$, pero cuatro grupos alcanzan solamente un TPR alrededor de $0{,}72$. Estos cuatro grupos justamente son los que tienen la mayor cantidad de parámetros (y por lo tanto la mayor cantidad de *features*), y esto podría indicar que nuestro WAF tiene dificultades con peticiones que tienen muchos parámetros.

En la segunda característica analizada mediante las pruebas, tratamos de cuantificar el impacto que nuestro WAF podría tener sobre el tiempo de respuesta de las aplicaciones web que protege. A pesar de que nuestra implementación puede ser optimizada, logramos que el retraso adicional en el tiempo de repuesta sea despreciable comparado con la latencia promedio del tráfico de red en Internet, por lo que el trabajo de detección de nuestro WAF no afectaría de forma notable el tiempo de respuesta de las aplicaciones protegidas.

En la tercera característica analizada mediante las pruebas, que trata del tiempo de entrenamiento de nuestro WAF, verificamos que esta duración depende de la cantidad de peticiones utilizadas para dicha fase de entrenamiento. Los números indican que se puede entrenar nuestra implementación con $100\,000$ peticiones en alrededor de 12 minutos. Esta rapidez le provee mucha flexibilidad a los administradores del WAF para poder planificar los momentos de entrenamiento. Recordamos que el entrenamiento solamente es necesario después de realizar cambios en las aplicaciones protegidas, y la eficacia de detección del sistema no debería verse afectada por la aparición de nuevos ataques.

Al final de la Sección V realizamos también una comparación de OCS-WAF con propuestas de otros trabajos del área. Para poder comparar los resultados, nos enfocamos en aquellos que utilizan el mismo conjunto de datos que nosotros empleamos para las pruebas. Analizamos propuestas que utilizan distintas herramientas, como por ejemplo árboles de decisión, modelos estadísticos y también el mismo clasificador One-Class SVM.

VI-B. Alcance de los objetivos

En esta parte presentamos los objetivos que nos habíamos propuesto para esta investigación y mostramos de qué manera logramos alcanzar los mismos a través del trabajo realizado. Empezamos describiendo nuestros logros para los objetivos específicos, para después cerrar con el objetivo general.

VI-B1. Objetivos específicos:

1. Diseñar procesos de extracción de características (*features*) específicamente para mensajes HTTP, basado en aportes de otros investigadores de la literatura.
   - Partiendo de los trabajos de los autores Kruegel y Vigna, y combinando sus aportes con trabajos de la literatura especializada, se diseñó procesos de extracción de *features* que pueden representar características de los mensajes HTTP mediante vectores de *features* numéricos. Estos procesos de extracción fueron descritos en la Sección II, mientras que en la Sección IV se mostraron los detalles de la implementación de los mismos. Las pruebas experimentales presentadas en la Sección V demuestran la utilidad de estos procesos diseñados para la tarea de detección de anomalías en mensajes HTTP.
2. Implementar un WAF basado en anomalías, utilizando los procesos de extracción de *features* diseñados junto con clasificadores One-Class SVM.
   - Se implementó un WAF sencillo, utilizando los procesos de extracción de *features* que fueron descritos en la Sección II, junto con los clasificadores One-Class SVM que fueron presentados en la Sección III. Los detalles de implementación del OCS-WAF fueron explicados en la Sección IV. Se incluyó la referencia al repositorio público que contiene el código fuente del OCS-WAF.
3. Evaluar la eficacia del WAF implementado en cuanto a la detección de mensajes HTTP anómalos.
   - Mediante las pruebas de la Sección V se mostró que el OCS-WAF es eficaz en la detección de mensajes

HTTP anómalos, considerando los conjuntos de datos de prueba utilizados en este trabajo.

4. Analizar la viabilidad de utilizar el WAF implementado para detección de ataques en tiempo real.
   - Utilizando la implementación sencilla, con las pruebas de la Sección V se mostró que el OCS-WAF no afectaría de forma notable el tiempo de respuesta de las aplicaciones web que están siendo protegidas por el mismo, posibilitando que pueda ser utilizado para detección de ataques en tiempo real.

VI-B2. Objetivo general:

- Detectar mensajes HTTP anómalos entre aplicaciones web y sus usuarios con el fin de mitigar los riesgos de ataques contra dichas aplicaciones, utilizando un WAF basado en clasificadores One-Class SVM.
  - Se construyó un WAF que usa características específicas de mensajes HTTP y clasificadores One-Class SVM para detectar mensajes anómalos. OCS-WAF puede ser colocado frente a múltiples aplicaciones web con el fin de protegerlas contra posibles ataques. De esta forma, en nuestra opinión, se ha logrado cumplir satisfactoriamente con todos los objetivos propuestos para este trabajo.

VI-C. Ideas para trabajos futuros

A pesar de concluir exitosamente nuestra investigación, reconocemos que aún hay varios aspectos que pueden ser mejorados o extendidos en trabajos futuros.

- Se podría realizar pruebas con OCS-WAF utilizando otros conjuntos de datos.
- Se podría explorar otras características de los mensajes HTTP para extender y mejorar nuestros procesos de extracción de *features*.
- Actualmente nuestra implementación solamente considera los cuerpos de peticiones que constan de pares de parámetros y valores estructurados de la misma forma que el *query string*. Se puede extender el OCS-WAF para incluir cuerpos de peticiones de otros formatos, por ejemplo datos binario, JSON, XML, entre otros.
- OCS-WAF solamente considera mensajes HTTP en la versión 1.1 del protocolo. En la versión 2 de este protocolo, dichos mensajes son enviados en formato binario [25] y se podría incluir una extensión a nuestra implementación para poder analizar también los mensajes HTTP/2.
- OCS-WAF no cuenta todavía con un método automático para la selección de los valores ν_i y γ_i. Se podría explorar métodos para realizar la selección óptima de estos parámetros, para posteriormente incorporar también ese mecanismo descubierto dentro de nuestra implementación.

REFERENCIAS

[1] J. Giménez, "Http-ws-ad: Detector de anomalías orientada a aplicaciones web y web services," Universidad Nacional de Asunción, 2015.

[2] W. Robertson, "Detecting and preventing attacks against web applications," Ph.D. dissertation, University of California, Santa Barbara, 2009.

[3] K. A. Scarfone and P. M. Mell, "Sp 800-94. guide to intrusion detection and prevention systems (idps)," National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2007.

[4] C. Torrano-Giménez, "Study of stochastic and machine learning techniques for anomaly-based web attack detection," Ph.D. dissertation, Universidad Carlos III de Madrid, 2015.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc 2616: Hypertext transfer protocol – http/1.1," RFC Editor, United States, Tech. Rep., 1999.

[6] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 2003.

[7] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010.

[8] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, 2016.

[9] C. C. Aggarwal, *Outlier Analysis*. Springer Publishing Company, Incorporated, 2013.

[10] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Irish Conference on Artificial Intelligence and Cognitive Science*. Springer Berlin Heidelberg, 2009.

[11] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, 2001.

[12] S. Khan and M. Madden, "One-class classification: taxonomy of study and review of techniques," *The Knowledge Engineering Review*, 2014.

[13] C. Kruegel, G. Vigna, and W. Robertson, "A multi-model approach to the detection of web-based attacks," *Computer Networks*, 2005.

[14] R. Dobrushin and V. Prelov, "Entropy - encyclopedia of mathematics," http://www.encyclopediaofmath.org/index.php?title=Entropy&oldid=15099, 2011, accessed: August-2017.

[15] H. T. Nguyen, C. Torrano-Gimenez, G. Alvarez, S. Petrović, and K. Franke, "Application of the generic feature selection measure in detection of web attacks," in *Computational Intelligence in Security for Information Systems*. Springer, Berlin, Heidelberg, 2011.

[16] K. Rieck, "Machine learning for application-layer intrusion detection," Ph.D. dissertation, Technischen Universität Berlin, 2009, accessible under: http://dx.doi.org/10.14279/depositonce-2199.

[17] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems," in *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE, 2006.

[18] E. Parhizkar and M. Abadi, "Oc-wad: A one-class classifier ensemble approach for anomaly detection in web traffic," in *2015 23rd Iranian Conference on Electrical Engineering (ICEE)*. IEEE, 2015.

[19] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*. ACM, 2013.

[20] P. Lagadec, "Cherryproxy - a filtering http proxy extensible in python," http://www.decalage.info/python/cherryproxy, 2011, accessed: July-2017.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, 2011.

[22] C. Torrano-Giménez, A. Pérez Villegas, and G. Álvarez Marañón, "Csic 2010 http data sets," http://www.isi.csic.es/dataset/, 2010, accessed: July-2017.

[23] C. Torrano-Giménez, A. Pérez, and G. Álvarez, "Csic torpeda 2012 http data sets," http://www.tic.itefi.csic.es/torpeda, 2012, accessed: July-2017.

[24] "Internet weather map," https://www.internetweathermap.com/map, 2017, accessed: August-2017.

[25] M. Belshe, R. Peon, and N. Thomson, "Rfc 7540: Hypertext transfer protocol version 2 – http/2," RFC Editor, United States, Tech. Rep., 2015.