Universidad Nacional De Asunción

Facultad Politécnica

Ingeniería en Informática

Trabajo Final de Grado

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

Autores:

Nico Epp

Ralf Funk

Tutor:

Prof. Cristian Cappo

San Lorenzo - Paraguay
Noviembre - 2017

Dedico este trabajo a mi esposa Joyce.

Nico Epp

Dedico este trabajo a mi esposa Sara.

Ralf Funk

Agradecimientos

Después de haber logrado concluir este trabajo de investigación, queremos expresar nuestra gratitud

- a nuestro Señor Jesús, por su guía en todo este camino y las fuerzas necesarias para concluir este proceso de formación.
- a nuestros familiares y amigos, por las innumerables palabras de aliento y el apoyo incondicional de siempre.
- a nuestro tutor, por la orientación y ayuda para la realización exitosa de esta investigación.
- a los profesores de nuestra facultad, por la dedicación al transmitirnos conocimientos y valores durante nuestros años en esta casa de estudios.

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.

Palabras clave: Sistemas de Detección de Intrusión (IDS), Web Application Firewall (WAF), ataques web, detección de anomalías, One-Class SVM

Abstract

Vulnerabilities in web applications pose a great risk because they can be exploited by malicious attackers through the Internet. Web Application Firewalls (WAF) can be placed in front of these applications to detect possible attacks, thus reducing the impact of these risks. In this work, we present OCS-WAF, a WAF that can be placed in front of web applications to analyze the incoming HTTP messages, in order to detect anomalous messages that could contain attacks. Our implementation, made with the *Python* programming language, uses One-Class SVM classifier for the detection, coupled with custom HTTP-specific feature extraction processes. OCS-WAF is trained with messages that represent the normal behavior of the protected applications, and later, during the detection phase, it can detect anomalous messages or attacks. With anomaly detection strategy, OCS-WAF only needs to be retrained when there are changes in the protected applications, but the discovery of new attacks does not require this retraining. Our detection efficacy tests show that OCS-WAF reaches an average TPR of 0,93, an average FPR of 0,03 and an average F_1-*score* of 0,95 for the public data sets that we used. The tests that we applied also show that the detection process of OCS-WAF should not have a noticeable effect on the response time of the protected applications. Besides, it can be trained with 100 000 normal messages in only a few minutes. Finally, the source code of OCS-WAF is available in our public repository under `https://github.com/nico-ralf-ii-fpuna/tfg`, so that others may reproduce our results and extend our work with further research.

Keywords: Intrusion Detection System (IDS), Web Application Firewall (WAF), web attacks, Anomaly Detection, One-Class SVM

Contenido

Lista de Figuras

Lista de Símbolos

B_i	lista de parámetros del cuerpo de las peticiones HTTP del grupo G_i
F_i	conjunto de vectores de *features* que representan las peticiones HTTP del grupo G_i
G	conjunto de grupos de peticiones HTTP agrupadas por método y URL
G_i	el grupo i de peticiones HTTP que corresponden a un mismo método y URL
M_i	matriz de *features* que representa las peticiones HTTP del grupo G_i utilizadas para entrenamiento
Q_i	lista de parámetros del *query string* de las peticiones HTTP del grupo G_i
γ_i	parámetro del *kernel* RBF que regula la dimensión de la región de influencia de cada muestra de entrenamiento para el grupo G_i
ν_i	parámetro del One-Class SVM que regula la fracción de muestras situadas al mismo lado del hiperplano que el origen para el grupo G_i
ρ_i	distancia al origen del hiperplano del One-Class SVM para el grupo G_i
$\vec{f_{ij}}$	vector de *features* que representa una petición HTTP del grupo G_i
$\vec{w_i}$	vector perpendicular al hiperplano del One-Class SVM para el grupo G_i
ξ_i	lista de valores de holgura para el One-Class SVM del grupo G_i
m	cantidad de *features* extraídos por cada valor analizado
n_i	cantidad de componentes de los vectores $\vec{f_{ij}}$ del conjunto F_i

Lista de Acrónimos

CSIC	Consejo Superior de Investigaciones Científicas de España
FPR	*False Positives Rate*
GB	*Gigabytes*
GHz	*Gigahertz*
HIDS	*Host-based Intrusion Detection System*
HTTP	*Hypertext Transfer Protocol*
IDS	*Intrusion Detection System*
IP	*Internet Protocol*
IPS	*Intrusion Prevention System*
MB	*Megabytes*
ML	*Machine Learning*
NIDS	*Network-based Intrusion Detection System*
OCC	*One-Class Classification*
OCS-WAF	*One-Class* SVM *Web Application Firewall*
One-Class SVM	*One-Class Support Vector Machine*
RAM	*Random Access Memory*
RBF	*Radial Basis Function kernel*

SVM	*Support Vector Machine*
TPR	*True Positives Rate*
URL	*Universal Resource Locator*
WAF	*Web Application Firewall*

Lista de Tablas

Capítulo 1

Introducción

En este capítulo introducimos el tema de este trabajo. Describimos el área de estudio que rodea nuestra investigación, mostramos la problemática que nos llevó a realizar este trabajo y presentamos los objetivos que nos hemos propuesto para el mismo.

1.1. Motivación

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 (Giménez, 2015). Muchas de las aplicaciones web actualmente no están construidas de acuerdo a las mejores prácticas de seguridad, posibilitando que dichas aplicaciones queden vulnerables a diferentes ataques. Esto se debe a la falta de consciencia sobre la importancia de la seguridad y en muchos casos también a una falta de tiempo, ya que se suele priorizar el desarrollo de funcionalidades por encima de la seguridad. Esta es la situación de aplicaciones existentes como también lo puede ser para aplicaciones futuras. Por lo tanto se necesitan soluciones para mitigar los riesgos presentes (Robertson, 2009).

En este trabajo nosotros investigamos sobre mecanismos externos especializados en la detección de ataques, con el fin de mitigar los riesgos creados por las vulnerabilidades presentes en las aplicaciones web.

Los sistemas de detección de intrusión (IDS - *Intrusion Detection System*) son programas o dispositivos especializados para monitorear las actividades en un sistema o en una red en busca de intrusiones no autorizadas o posibles ataques. Las respuestas frente a intrusiones pueden

ser variadas, como por ejemplo el envío de mensajes de alerta o incluso medidas concretas de mitigación y contención de los posibles ataques. En este último caso se puede hablar también más específicamente de sistemas de prevención de intrusión (IPS - *Intrusion Prevention System*) (Scarfone and Mell, 2007).

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 (Torrano-Giménez, 2015). Como las aplicaciones web utilizan mayormente el protocolo HTTP (*Hypertext Transfer Protocol*) (Fielding et al., 1999) para sus comunicaciones, se necesita un IDS que pueda monitorear el tráfico HTTP, analizando los mensajes HTTP enviados y recibidos a través de las conexiones de red. En este caso se puede hablar más específicamente de cortafuegos para aplicaciones web (WAF - *Web Application Firewall*) (Torrano-Giménez, 2015).

En este trabajo presentamos OCS-WAF, un sistema de detección de ataques contra aplicaciones web. Cabe mencionar que debido a que WAF es un término más específico que IDS, muchos de los conceptos expuestos en este trabajo aplican a ambos términos, pero nosotros nos enfocamos en el término WAF.

Los WAF pueden utilizar dos métodos distintos para la detección de intrusiones. Una forma puede ser la búsqueda de patrones de ataques conocidos, llamado también método basado en firmas de ataques (*signature-based detection*). Otro método empleado es la búsqueda de anomalías en los mensajes HTTP con respecto al tráfico normal, ya que estas desviaciones pueden indicar ataques (*anomaly-based detection*) (Torrano-Giménez, 2015).

Para que un WAF pueda utilizar eficazmente el método por firmas, es necesario que el mismo mantenga una lista actualizada de las firmas de los ataques conocidos. La lista de firmas de ataques descubiertos crece constantemente y probablemente nunca deje de crecer. Durante el análisis de los 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 para este proceso de detección (Kruegel and Vigna, 2003).

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. Se basa en la premisa de que los ataques se diferencian en alguna forma de los mensajes normales. En adelante usamos el término anomalía para referirnos a los ataques, para ser consistentes con la literatura relacionada. 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 (Kruegel and Vigna, 2003). 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 (Kruegel and Vigna, 2003).

A pesar de esta importante ventaja, los WAFs basados en anomalías no son tan comunes como aquellos basados en firmas (Sommer and Paxson, 2010) (Kruegel and Vigna, 2003). Esto se debe en parte a que suele ser más complicado construir modelos significativos para diferenciar mensajes normales de anómalos y, como consecuencia, hay menos posibilidades de detectar eficazmente las anomalías. De esta manera los métodos por anomalías corren el peligro de caer en extremos. Por un lado, si se concentran en detectar todas las anomalías (que son las muestras positivas), pueden marcar equivocadamente mensajes normales como anómalos (genera más errores de falsos positivos). Por otro lado, si priorizan no bloquear ningún mensajes normal (que son las muestras negativas), puede que muchas anomalías no sean detectadas (genera más errores de falsos negativos) (Torrano-Giménez, 2015).

El WAF que presentamos en este trabajo emplea detección de anomalías en los mensajes HTTP, buscando mejorar algunas de las propuestas que ya han sido presentadas por otros investigadores, como por ejemplo (Kruegel and Vigna, 2003), (Giménez, 2015), (Giménez and Cappo, 2015) y (Torrano-Giménez, 2015).

Para la detección de anomalías se puede utilizar varias estrategias. Una opción es emplear herramientas estadísticas, como podemos ver en los trabajos (Kruegel and Vigna, 2003), (Giménez, 2015) y (Torrano-Giménez, 2015). Otra opción son herramientas del área de aprendizaje de máquinas (ML - *Machine Learning*) para tratar de detectar las anomalías; podemos ver esto en los trabajos (Sommer and Paxson, 2010), (Buczak and Guven, 2016), (Parhizkar and Abadi, 2015) y (Torrano-Giménez, 2015).

Las herramientas de ML han sido empleadas con éxito en varias áreas de la computación, como por ejemplo, en sistemas de recomendación, clasificación de imágenes, reconocimiento óptico de caracteres, entre otros (Torrano-Giménez, 2015).

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 (Torrano-Giménez, 2015). 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 (Aggarwal, 2013).

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*) (Khan and Madden, 2009). 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. Esto provee robustez al clasificador frente a la aparición de novedosas muestras que no pertenecen a la clase conocida. Esta estrategia ha sido utilizada con éxito en varias áreas, como detección de spam, reconocimiento de rostros, detección de fallas en maquinarias, entre otros (Khan and Madden, 2014). 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 o ataques son las muestras positivas, que no pertenecerán a la clase conocida de los mensajes normales (que son las muestras negativas).

Este trabajo presenta OCS-WAF, que emplea OCC con herramientas de ML para detectar mensajes anómalos. Con este abordaje solamente se necesita realizar una vez el entrenamiento con mensajes normales y, mientras no cambien las aplicaciones protegidas, no debería ser necesario volver a entrenarlo, aún con la aparición de nuevos ataques.

Los algoritmos o herramientas utilizados en ML son muy diversos, como por ejemplo árboles de decisiones (Torrano-Giménez, 2015), redes neuronales (Corchado and Herrero, 2011), algoritmos genéticos (Abadeh et al., 2011), entre otros (Torrano-Giménez, 2015). Una de estas herramientas, que ha sido utilizada con mucho éxito en las tareas de clasificación, es la máquina de vectores de soporte (SVM - *Support Vector Machine*). Una versión modificada del SVM ha sido propuesta como una de varias alternativas para afrontar tareas de OCC (Schölkopf et al., 2001). Varios investigadores ya han empleado exitosamente este clasificador One-Class SVM en problemas de diversas áreas, como por ejemplo en clasificación de textos y rostros, detección de spam, detección de fallas en máquinas, entre otros (Khan and Madden, 2014).

El detector OCS-WAF presentado en este trabajo utiliza clasificadores One-Class SVM para detectar mensajes HTTP anómalos.

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 (Kruegel and Vigna, 2003). Además se debe expresar esas características en un formato procesable para las herramientas de detección. La mayoría de las herramientas de ML, incluyendo el clasificador One-Class SVM, no pueden trabajar con los datos crudos y necesitan un paso de preprocesamiento de datos. 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 (Torrano-Giménez, 2015). 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 (Kruegel and Vigna, 2003) y (Kruegel et al., 2005), 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*. Por ejemplo, el primer valor del vector puede indicar la cantidad de caracteres del mensaje HTTP, el segundo la cantidad de dígitos presentes y el tercero puede representar la longitud de todo el mensaje. 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 (Torrano-Giménez, 2015).

En este trabajo presentamos OCS-WAF, que utiliza conocimiento experto sobre los mensajes HTTP para extraer características útiles para la detección de anomalías. Debido a que trabajamos con clasificadores One-Class SVM, en este trabajo utilizamos el término *feature* para referirnos a las características extraídas de los mensajes, para ser consistentes con la terminología del área de ML.

1.2. Justificación

Resumiendo lo expuesto anteriormente, en este trabajo combinamos tres áreas de estudio para proponer una solución a la problemática descrita. Dichas áreas se pueden observar también en la Figura 1.1.

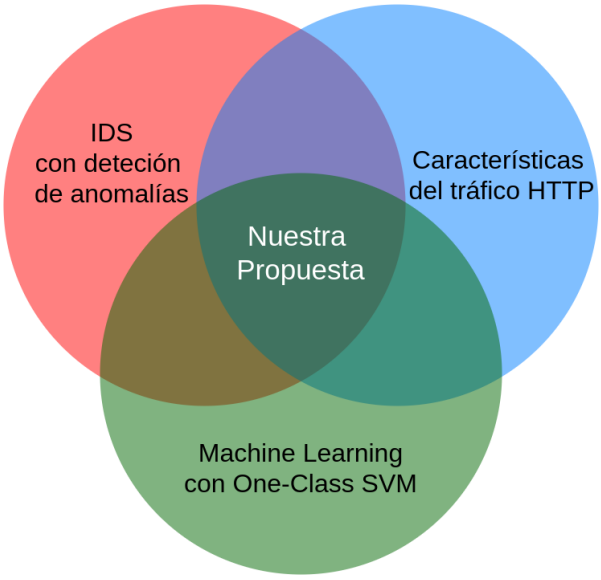

Figura 1.1: Diagrama de las áreas de estudio de la investigación.

- IDS con detección de anomalías: OCS-WAF busca detectar posibles ataques al reconocerlos como mensajes anómalos. Utilizamos el método de detección de anomalías debido a sus ventajas que fueron mencionadas en la sección anterior.
- Características de los mensajes HTTP: OCS-WAF utiliza conocimiento experto sobre la estructura de los mensajes HTTP, que puede ayudar para diferenciar los mensajes normales de las anomalías o ataques, como se puede observar en los trabajos (Kruegel and Vigna, 2003) y (Kruegel et al., 2005).

- ML con One-Class SVM: la detección de mensajes anómalos se puede abordar como un problema de OCC, y para ese tipo de problemas el clasificador en cuestión ha sido empleado exitosamente por otros investigadores, obteniendo buenos resultados en la clasificación (Khan and Madden, 2014).

En los trabajos de Kruegel y Vigna se combina las dos primeras áreas descritas anteriormente, IDS y características de los mensajes HTTP. Sus aportes ya fueron utilizados y extendidos en varios trabajos en años subsecuentes, pero nosotros no hemos encontrado una investigación que combine dichos aportes con el ya mencionado clasificador One-Class SVM.

En nuestra opinión, un WAF que combina los aportes de Kruegel y Vigna con este clasificador puede ser de gran utilidad para la detección de posibles ataques y de esta forma brindar una herramienta para mayor protección de las aplicaciones web.

1.3. Objetivos

1.3.1. Objetivo general

Detectar mensajes HTTP anómalos entre las 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.

1.3.2. Objetivos específicos

1. Diseñar procesos de extracción de características (*features*) específicamente para mensajes HTTP, basado en aportes de otros investigadores de la literatura.

2. Implementar un WAF basado en anomalías, utilizando los procesos de extracción de *features* diseñados junto con clasificadores One-Class SVM.

3. Evaluar la eficacia del WAF implementado en cuanto a la detección de mensajes HTTP anómalos.

4. Analizar la viabilidad de utilizar el WAF implementado para detección de ataques en tiempo real.

1.4. Organización del trabajo

Este trabajo está organizado de la siguiente manera: en el Capítulo 2 entramos en más detalles sobre cómo utilizamos las características de los mensajes HTTP en nuestros procesos de extracción de *features*. Luego, en el Capítulo 3 describimos con más detalles el clasificador One-Class SVM, mostrando de qué manera lo empleamos para la detección de mensajes anómalos. El Capítulo 4 contiene la presentación del WAF que construimos en el marco de este trabajo, describiendo varios detalles de la implementación y mostrando de qué manera utilizamos los componentes descritos en los capítulos 2 y 3. Después de eso, en el Capítulo 5, describimos las pruebas realizadas con nuestra implementación y exponemos los resultados que hemos obtenido en cada una de ellas. Finalizamos el presente trabajo con nuestras conclusiones en el Capítulo 6, agregando también algunas ideas que pueden ser el punto de partida para trabajos futuros.

Capítulo 2

Extracción de *features* en el contexto HTTP

La extracción de *features* consiste en obtener características representativas de los datos originales (Torrano-Giménez, 2015). En nuestro caso, las características de los mensajes HTTP deben ser expresadas por medio de vectores numéricos para que los clasificadores One-Class SVM del OCS-WAF puedan reconocer dichas características. En (Rieck, 2009) se habla también de mapas de *features* para referirse a esta extracción de características; se presenta este proceso como funciones que mapean el dominio de mensajes HTTP a un espacio vectorial $\mathbb{R}^n$ de números reales, donde $1 \leq n \leq \infty$.

En este capítulo, en primer lugar damos algunas explicaciones sobre la estructura de los mensajes HTTP, luego presentamos los aportes de los trabajos de Kruegel y Vigna sobre las características que utilizaron, y finalmente describimos los procesos de extracción de *features* que utiliza OCS-WAF para realizar el mapeo de mensajes a vectores numéricos.

2.1. Estructura de mensajes HTTP

El detector OCS-WAF que presentamos en este trabajo analiza mensajes HTTP y en esta sección introducimos los conceptos necesarios sobre este protocolo de comunicación. Nosotros solamente nos enfocamos en el protocolo HTTP en su versión 1.1, en el cual los mensajes son enviados en formato de texto (Fielding et al., 1999).

Además, OCS-WAF analiza solamente los mensajes HTTP que sean peticiones, ignorando los mensajes que sean respuestas. Debido a que las peticiones son aquellos mensajes que llegan a las aplicaciones web desde la Internet, estos pueden contener ataques; en cambio, los mensajes

de respuesta son generadas por las aplicaciones protegidas por el WAF y por lo tanto hay menos riesgos de seguridad. Obviando el análisis de las respuestas reducimos el procesamiento necesario sin aumentar de forma significativa el riesgo de ataques.

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 (Fielding et al., 1999). El método debe ser uno de los valores definidos para el protocolo; nuestro WAF se enfoca solamente en peticiones con los métodos GET y POST, ya que estos son los más utilizados. La URL está compuesta por el indicador de protocolo, la dirección del *host*, el puerto y la ruta del recurso pedido (*path*). Las cabeceras son opcionales y permiten enviar información adicional sobre la petición. Cada cabecera consiste en un nombre y su valor, separados por el signo de dos puntos (:). Las cabeceras van delimitadas por los caracteres de nueva linea (CRLF).

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 la lista de pares ordenados $\{(p_1, v_1), (p_2, v_2), \ldots, (p_n, v_n)\}$. Estos parámetros son utilizados, por ejemplo, para pasar criterios de filtrado y ordenación de datos a las aplicaciones.

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*. De esta manera, el cuerpo de una petición puede ser expresado como la lista $\{(p_1, v_1), (p_2, v_2), \ldots, (p_n, v_n)\}$. En esta investigación, no analizamos cuerpos de peticiones con contenido en otros formatos, por ejemplo aquellos que tengan datos binarios. Los cuerpos de peticiones POST pueden ser utilizados, por ejemplo, para enviar datos de formularios completados por usuarios a aplicaciones web.

En la Figura 2.1 se puede observar la estructura de dos peticiones de ejemplo. La primera es una petición con el método GET. La misma tiene un *query string* con los dos parámetros *email* y *full* con sus respectivos valores. Esta primera petición no tiene cuerpo. La segunda petición tiene el método POST y no cuenta con un *query string*, pero en el cuerpo contiene los parámetros *modo* y *precio* con sus valores. Ambas peticiones contienen dos cabeceras que son *Accept-Charset* y *Accept-Language* con sus valores. Cabe mencionar que los caracteres de

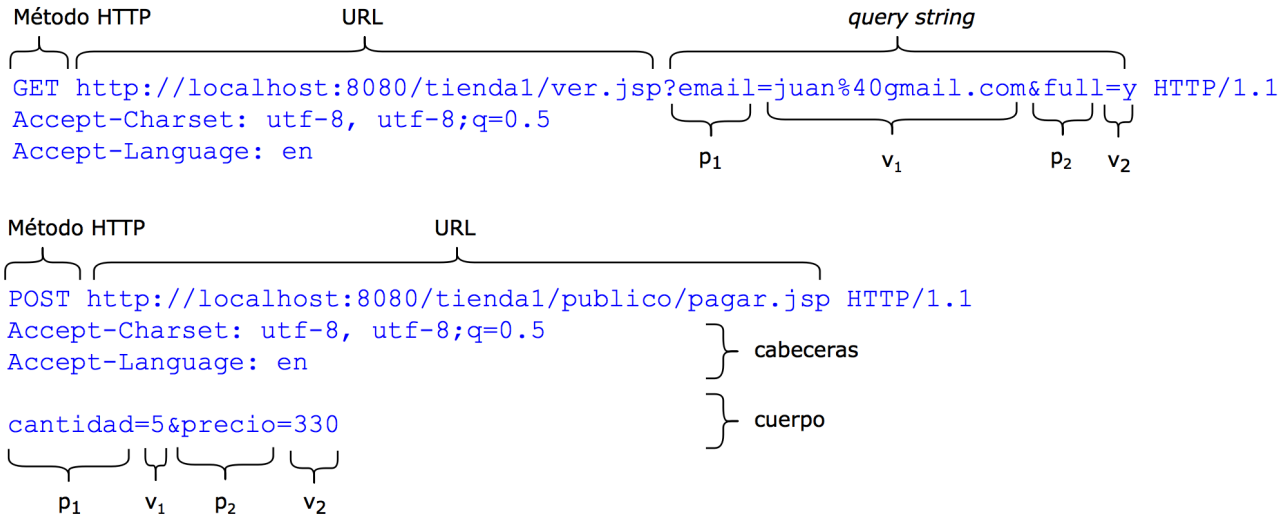

Figura 2.1: Diagrama de estructura de dos peticiones HTTP, una con método GET y otra con método POST.

la petición que tienen significado especial en el contexto del mensaje deben ser reemplazados por un código hexadecimal precedido por un símbolo de porcentaje (%), un proceso conocido como *percent encoding* (Berners-Lee et al., 2005). Este reemplazo se puede observar en el valor del parámetro *email* de la primera petición de la figura mencionada, donde el símbolo @ queda reemplazado por el código %40.

2.2. Características HTTP según Kruegel y Vigna

Los autores Kruegel y Vigna presentaron varias ideas relacionadas a la detección de anomalías en peticiones HTTP. En los trabajos (Kruegel and Vigna, 2003) y (Kruegel et al., 2005) utilizan conocimiento experto sobre mensajes HTTP, específicamente sobre las peticiones, para construir modelos de anomalías durante una fase de entrenamiento y luego, durante una fase de detección, emplean esos modelos para obtener una detección eficaz. Nosotros basamos nuestra propuesta en dos aspectos de sus aportes: en primer lugar usamos el análisis de valores de parámetros del *query string* que ellos proponen y en segundo lugar empleamos algunas de las características que ellos utilizan en sus modelos de anomalías.

Cabe mencionar que Kruegel y Vigna utilizan métodos estadísticos para la detección de anomalías y no herramientas de ML como nosotros. Debido a eso, ellos no utilizan vectores de *features*. Aun así, utilizamos los dos aspectos de sus aportes mencionados anteriormente para mejorar nuestros procesos de extracción de *features*, apuntando a mejorar la detección de

peticiones anómalas del OCS-WAF. Adicionalmente, Kruegel y Vigna realizan sus tareas de detección de anomalías en base a archivos *logs* de servidores HTTP, pero no trabajan sobre tráfico en tiempo real. Como en esos *logs* solamente había peticiones con el método GET, sus trabajos tratan únicamente de ese tipo de peticiones.

2.2.1. Análisis de valores de parámetros

Los autores Kruegel y Vigna aplican sus modelos de anomalías sobre el *query string* de peticiones GET y además sobre los valores individuales de los parámetros de la misma. En la fase de detección, sus modelos obtienen una probabilidad de que el valor analizado pertenezca al conjuntos de valores vistos durante el entrenamiento.

Durante la fase de entrenamiento, en primer lugar las peticiones HTTP son agrupadas por método y URL; los autores utilizaron solamente peticiones GET y así en la práctica su agrupación fue por URL únicamente. Esta agrupación se realiza ya 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. De esta manera, se obtiene modelos de anomalías independientes y más precisos dentro de cada grupo. Después de agrupar, se realiza la extracción de los parámetros que aparecen en el *query string* de las peticiones dentro de cada grupo. Luego se construye modelos de anomalías por cada uno de los parámetros extraídos, utilizando los valores que dichos parámetros tienen en las peticiones del conjunto de entrenamiento. Por ejemplo, se analiza la longitud de todos los valores de un parámetro, construyendo de esta forma un modelo de anomalía que utiliza promedio y desviación estándar de las longitudes para definir a partir de qué longitud un valor de ese parámetro específico será considerado anómalo. Además, se tiene modelos aplicados al *query string* completo, como por ejemplo el análisis de presencia de parámetros.

Durante la fase de detección, el sistema propuesto por los autores mencionados analiza las peticiones nuevas con los modelos construidos previamente. Primeramente se busca el grupo al cual pertenece la petición, luego se obtiene la lista de parámetros vistos durante el entrenamiento para ese grupo y luego se extraen todos los valores que esos parámetros tienen en la petición en cuestión. Después se aplican los modelos sobre los valores, obteniendo resultados que indican la probabilidad de que los nuevos valores sean normales. Así, por cada parámetro de la petición se tendrá varios resultados calculados por los distintos modelos, de los cuales se obtiene una probabilidad final mediante una suma ponderada. Esta probabilidad final es comparada con un umbral (*threshold*) establecido previamente para decidir si la petición analizada es normal o anómala.

En esta parte cabe resaltar que el análisis hecho por estos autores utiliza una cantidad variable de modelos en cada grupo de peticiones, ya que cada grupo puede tener una cantidad distinta de parámetros; esa cantidad de parámetros observados durante el entrenamiento determina la cantidad de modelos que se tendrá. Este abordaje es diferente a otros trabajos relacionados al área de IDS en donde se utiliza una cantidad fija de características de los mensajes; podemos ver un ejemplo de esto en (Nguyen et al., 2011), donde se utiliza 30 *features* de peticiones HTTP para los distintos clasificadores empleados.

2.2.2. Características analizadas de los mensajes

Kruegel y Vigna analizan un total de nueve características distintas en sus trabajos (Kruegel and Vigna, 2003) y (Kruegel et al., 2005), a partir de las cuales construyen sus modelos de anomalías; estas características pueden ser observadas en la Tabla 2.1.

Los autores detallan también los tipos de ataques que se busca detectar con cada uno de sus modelos de anomalías. Entre los ataques detectados se encuentran, por ejemplo, ataques de *buffer overflow*, *directory transversal*, *cross-site scripting* y también *SQL injection*.

Características analizadas por Kruegel y Vigna	Característica utilizada en OCS-WAF
Longitud	Sí
Distribución de caracteres	Sí
Inferencia de estructura	-
Análisis de *tokens*	-
Presencia o ausencia de parámetros	-
Orden de parámetros	-
Frecuencia de invocación	-
Tiempo entre invocaciones	-
Orden de invocación	-

Tabla 2.1: Características de mensajes HTTP utilizadas por Kruegel y Vigna en sus modelos de anomalías, indicando también cuales de estas características son utilizadas en OCS-WAF.

En nuestra opinión, no todas las características presentadas son aplicables a nuestro caso con herramientas de ML. De esa forma, nosotros empleamos solamente dos de las características de la tabla mencionada para construir los procesos de extracción de *features* del OCS-WAF; estos procesos son presentados en la siguiente sección.

2.3. Nuestros procesos de extracción de *features*

En esta sección presentamos los procesos que realizan la extracción de *features* en OCS-WAF. Primeramente, describimos la agrupación de peticiones que realizamos, luego explicamos el análisis de valores de parámetros que hacemos y después detallamos los *features* que extraemos de cada uno de los valores.

A modo de ilustrar mejor los conceptos de esta sección, utilizamos un conjunto de peticiones de ejemplo sobre el que aplicamos nuestros procesos de extracción que presentamos a continuación. En la Tabla 2.2 se pueden observar dichas peticiones de ejemplo, que son 20 peticiones GET que fueron hechas a una misma URL. De este conjunto, las primeras 15 peticiones son normales y las otras 5 contienen alguna anomalía o ataque. Además, las primeras 10 peticiones normales serán utilizadas para la fase de entrenamiento, mientras que las demás normales y las anomalías recién serán utilizadas para la fase de detección.

2.3.1. Agrupación de peticiones

Basado en los trabajos de Kruegel y Vigna mencionados en la sección anterior, OCS-WAF también realiza la agrupación de las peticiones HTTP por método y URL. De esta forma, a partir de las peticiones utilizadas para el entrenamiento, OCS-WAF 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 el contexto de otro grupo G_2.

En el caso de nuestras 20 peticiones de ejemplo, todas ellas fueron hechas con el método GET a una sola URL. Por lo tanto, dichas peticiones pertenecen al mismo grupo. Cabe volver a mencionar que para el entrenamiento utilizamos las 10 primeras peticiones normales.

2.3.2. Extracción de valores de los parámetros del *query string* y cuerpo de las peticiones

Con la agrupación realizada, partimos de la base de que las peticiones dentro de un grupo G_i presentan gran similitud entre ellas. Específicamente, se espera que esas peticiones tengan los mismos parámetros en *query string* y cuerpo, o que por lo menos no haya mucha variación. Por

ID	Clase	Grupo	Petición completa
1	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=talmadge%40movintel.uz HTTP/1.1`
2	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tarangelo%40accesoabierto.st HTTP/1.1`
3	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tarasova_dabo%40comerciosdeaspe.ly HTTP/1.1`
4	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tazaki_waggenheim%40telelujo.an HTTP/1.1`
5	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tati%40callalabocaatiaherminia.travel HTTP/1.1`
6	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tauber5%40lamolahotel.bb&full=yes HTTP/1.1`
7	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tayo%4013horas.tn HTTP/1.1`
8	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tanimoto_hassen%40autobuses-hibridos.tw HTTP/1.1`
9	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tarp%40redstarz.asia&full=yes HTTP/1.1`
10	normal	entrenamiento	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=teje-lipman%40hojainformativas.sa HTTP/1.1`
11	normal	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=taneda%40ajuntamentdebcn20.be HTTP/1.1`
12	normal	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tannen%40getgold.ua HTTP/1.1`
13	normal	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tchjan-lian.quadflieg%40comerciosdeaspe.se HTTP/1.1`
14	normal	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tarver%40callofduty6.lb&full=yes HTTP/1.1`
15	normal	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tay%40autobuses-hibridos.ps HTTP/1.1`
16	anomalía	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=%27%3B+DROP+TABLE+user%3B+SELECT+*+FROM+p+WHERE+name+LIKE+%27%25 HTTP/1.1`
17	anomalía	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=malformed1email2address3domain4 HTTP/1.1`
18	anomalía	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=tayo_1%4013horas.tn&full=12345&show=no HTTP/1.1`
19	anomalía	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=%3C%21--%23exec+cmd%3D%22cat+%2Fetc%2Fpasswd+%3E+somefile%22+--%3E HTTP/1.1`
20	anomalía	detección	`GET http://localhost:8080/tienda1/miembros/ver.jsp?email=%40cr%40anh%40am%404horas.ki&full=yes HTTP/1.1`

Tabla 2.2: Conjunto de 20 peticiones HTTP de ejemplo utilizadas para ilustrar el funcionamiento de OCS-WAF en este trabajo.

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.

Como ya se mencionó en la sección anterior, Kruegel y Vigna utilizan solamente peticiones GET en sus trabajos. OCS-WAF incluye el análisis de parámetros del cuerpo de las peticiones POST. Cabe mencionar que estamos conscientes del riesgo de seguridad que puede presentar el hecho de que se analice el cuerpo de peticiones POST, ya que los mismos pueden contener información confidencial, como por ejemplo contraseñas de acceso de usuarios. Para ajustarse a las necesidades de las distintas situaciones, OCS-WAF puede ser configurado para excluir ciertos métodos HTTP o ciertas URLs del análisis que realiza. Estas configuraciones quedan a cargo de los administradores del OCS-WAF.

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. En nuestras 10 peticiones de ejemplo para el entrenamiento, Q_i contiene los dos parámetros *email* y *full*, pero B_i queda vacío porque las peticiones no tienen cuerpo con 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 sección.

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 uno de los grupo de peticiones, representada

por n_i, indica la cantidad de componentes de cada $\vec{f_{ij}} \in F_i$ y esta dimensión está dada por la expresión de la Ecuación 2.1.

$$n_i = m \times (1 + |Q_i| + |B_i|) \tag{2.1}$$

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 de componentes n_i puede ser distinta en cada grupo G_i, ya que depende de la cantidad de parámetros que se encuentran en las peticiones de esos grupos. Para nuestras peticiones de ejemplo, $|Q_i| = 2$ y $|B_i| = 0$, de manera que $n_i = m \times 3$. Así, cada petición será representada por vectores de $m \times 3$ componentes. Recalcamos que los m *features* serán presentados más adelante en esta sección.

En la fase de detección, para cada petición a analizar se construye un vector de *features* que las represente. Se respeta el orden de los parámetros en Q_i y B_i según el grupo G_i al que pertenecen las peticiones. 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. Este es el caso de nuestra petición de ejemplo número 18, que tiene el parámetro *show* que no aparece en el entrenamiento. En cambio, si un parámetro visto durante la fase de entrenamiento no aparece en nuevas peticiones, los vectores de *features* llevarán 0 en todos los componentes que correspondan a ese parámetro. En nuestros ejemplos, el parámetro *full* solamente aparece en algunas de las peticiones y sus componentes correspondientes llevarán un 0 cuando no está presente. De esta forma, los vectores de *features* siempre tendrán la dimensión n_i que corresponde a su grupo.

2.3.3. *Features* extraídos de cada valor

La petición HTTP completa y también cada uno de los valores de los parámetros del *query string* y del cuerpo son analizados y mapeados a distintos componentes de los vectores $\vec{f_{ij}}$. Como ya fue mencionado, de cada uno de estos valores se extraen m *features*. Dentro de nuestros procesos de extracción de *features* utilizamos 10 números para representar cada valor, resultando en $m = 10$. En la Tabla 2.3 se detallan estos 10 *features* que utilizamos, los cuales serán explicados más adelante en este capítulo.

Para la extracción de esos 10 *features* de cada valor, utilizamos tres funciones que retornan uno o más números cada una; específicamente se tiene la primera función que analiza la distribución de caracteres, la segunda función que calcula la entropía del valor y por último la

tercera función que analiza la cantidad de caracteres. Estas funciones son presentadas en detalle a continuación. Cabe mencionar que los valores son analizados después revertir el *percent encoding*, es decir, después de sustituir los códigos hexadecimales precedidos por porcentaje con sus símbolos correspondientes.

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

Tabla 2.3: Lista de 10 *features* extraídos por nuestros procesos de extracción de *features* de cada valor analizado.

2.3.3.1. 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 aparición de los valores individuales, sino que se toma en cuenta la relación entre las frecuencias de los caracteres (Kruegel and Vigna, 2003).

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 poseer casi ninguna variación entre las frecuencias.

En la Tabla 2.4 se puede ver la distribución de caracteres de un valor de ejemplo, *empleado@empresa.com*. Este valor tiene una longitud de 20 caracteres, con 12 caracteres diferentes. Cada carácter diferente tiene su propia columna, junto con la frecuencia relativa correspondiente. Las posiciones en la lista de frecuencias inician en 0.

Posición	0	1	2	3	4	5	6	7	8	9	10	11
Carácter	e	m	a	o	p	c	d	l	r	s	.	@
Frecuencia	0,2	0,15	0,1	0,1	0,1	0,05	0,05	0,05	0,05	0,05	0,05	0,05

Tabla 2.4: Distribución de caracteres del valor *empleado@empresa.com*.

En este punto, los autores Kruegel y Vigna proponen agrupar las frecuencias relativas en intervalos o *bins* de distintos tamaños. Nosotros utilizamos cinco intervalos para esta agrupación de frecuencias. La Tabla 2.5 se puede observar los intervalos, las posiciones de la distribución que son sumadas en cada intervalo y también muestra las sumas obtenidas con el valor de ejemplo que se mencionó.

	Intervalo 0	Intervalo 1	Intervalo 2	Intervalo 3	Intervalo 4
Posiciones agrupadas	0	1 y 2	3, 4 y 5	6, 7, 8 y 9	10 al último
Suma de frecuencias	0,2	0,25	0,25	0,2	0,1

Tabla 2.5: Agrupación de la distribución de caracteres para el valor de ejemplo *empleado@empresa.com*.

Por otro lado, en la Tabla 2.6 podemos ver los intervalos de un valor que contiene un ataque de *buffer overflow* que consta de muchas repeticiones de un solo carácter. Se puede observar que el ataque contiene un número 1,0 en el intervalo 0, que es mucho más elevado que los números en los demás intervalos, que todos contienen 0,0. Esta caída abrupta no se observa con el valor de ejemplo normal, y de esta forma se puede detectar la anomalía.

	Intervalo 0	Intervalo 1	Intervalo 2	Intervalo 3	Intervalo 4
Posiciones agrupadas	0	1 y 2	3, 4 y 5	6, 7, 8 y 9	10 al último
Suma de frecuencias	1,0	0,0	0,0	0,0	0,0

Tabla 2.6: Agrupación de la distribución de caracteres de un valor de ejemplo que contiene un ataque de *buffer overflow*.

A partir de este punto los autores mencionados obtienen una distribución promedio en la fase de entrenamiento y aplican el método estadístico χ^2 de Pearson (Nikulin, 2012) para comparar ese promedio con las distribuciones obtenidas de los valores en la fase de detección. En cambio, nuestra función que analiza la distribución de caracteres retorna directamente las cinco sumas

obtenidas en los cinco intervalos para cada valor. Esos números 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.

En la Tabla 2.7 se puede observar los *features* de distribución de caracteres extraídos de nuestras peticiones de ejemplo. Las primeras 10 peticiones fueron utilizadas para el entrenamiento, mientras que las demás peticiones fueron analizadas con los parámetros encontrados en entrenamiento. Cada fila de la tabla representa 15 componentes del vector de *features* de la petición correspondiente. Mostramos los números redondeados a dos dígitos de precisión, a pesar de que nuestra implementación utiliza todos los decimales disponibles.

En esta parte queda también ilustrado el comportamiento de nuestros procesos de extracción de *features* frente a la aparición de parámetros en la fase de detección que no se observaron durante la fase de entrenamiento. Se puede notar que la petición de ejemplo número 18 de la Tabla 2.2 tiene el parámetro *show*, pero como este parámetro no aparece entre la peticiones de entrenamiento, el mismo no tiene sus propios componentes en el vector de *features* (y tampoco sus propias columnas en la Tabla 2.7 en cuestión).

ID	Petición completa					Parámetro *email*					Parámetro *full*				
1	0,07	0,14	0,17	0,16	0,47	0,10	0,20	0,25	0,20	0,25	0,00	0,00	0,00	0,00	0,00
2	0,07	0,15	0,18	0,16	0,44	0,15	0,23	0,27	0,19	0,15	0,00	0,00	0,00	0,00	0,00
3	0,08	0,14	0,17	0,16	0,45	0,16	0,22	0,22	0,16	0,25	0,00	0,00	0,00	0,00	0,00
4	0,08	0,13	0,16	0,14	0,47	0,14	0,21	0,21	0,17	0,28	0,00	0,00	0,00	0,00	0,00
5	0,12	0,15	0,18	0,15	0,41	0,26	0,23	0,23	0,14	0,14	0,00	0,00	0,00	0,00	0,00
6	0,08	0,13	0,17	0,15	0,46	0,14	0,27	0,27	0,18	0,14	0,33	0,67	0,00	0,00	0,00
7	0,07	0,13	0,16	0,16	0,48	0,13	0,27	0,20	0,27	0,13	0,00	0,00	0,00	0,00	0,00
8	0,08	0,14	0,17	0,16	0,45	0,14	0,22	0,22	0,22	0,22	0,00	0,00	0,00	0,00	0,00
9	0,07	0,13	0,18	0,16	0,46	0,22	0,28	0,22	0,22	0,06	0,33	0,67	0,00	0,00	0,00
10	0,08	0,12	0,17	0,17	0,45	0,16	0,16	0,19	0,23	0,26	0,00	0,00	0,00	0,00	0,00
11	0,08	0,15	0,16	0,14	0,47	0,15	0,30	0,26	0,15	0,15	0,00	0,00	0,00	0,00	0,00
12	0,07	0,14	0,15	0,15	0,48	0,18	0,24	0,29	0,24	0,06	0,00	0,00	0,00	0,00	0,00
13	0,08	0,12	0,16	0,17	0,47	0,12	0,17	0,20	0,20	0,30	0,00	0,00	0,00	0,00	0,00
14	0,08	0,12	0,15	0,14	0,50	0,14	0,19	0,19	0,19	0,29	0,33	0,67	0,00	0,00	0,00
15	0,08	0,13	0,16	0,15	0,48	0,16	0,16	0,24	0,20	0,24	0,00	0,00	0,00	0,00	0,00
16	0,08	0,10	0,12	0,12	0,57	0,20	0,17	0,13	0,13	0,37	0,00	0,00	0,00	0,00	0,00
17	0,07	0,14	0,16	0,20	0,42	0,13	0,26	0,23	0,23	0,16	0,00	0,00	0,00	0,00	0,00
18	0,07	0,12	0,15	0,15	0,51	0,12	0,24	0,24	0,24	0,18	0,20	0,40	0,40	0,00	0,00
19	0,08	0,13	0,14	0,15	0,51	0,11	0,20	0,20	0,17	0,33	0,00	0,00	0,00	0,00	0,00
20	0,06	0,12	0,15	0,17	0,51	0,20	0,25	0,20	0,20	0,15	0,33	0,67	0,00	0,00	0,00

Tabla 2.7: Distribución de caracteres de nuestras 20 peticiones de ejemplo.

2.3.3.2. 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 2.2 muestra la fórmula utilizada para calcular la entropía, propuesta por Claude Shannon hace varias décadas (Dobrushin and Prelov, 2011).

$$H(x) = -\sum_{i=1}^{n} \left(\frac{c_i}{n} \times \log_2 \frac{c_i}{n} \right) \tag{2.2}$$

En la ecuación mencionada, el símbolo x representa el valor del cual se calcula la entropía (que en nuestro caso son cadenas de caracteres), el símbolo n es la cantidad de caracteres distintos que se puede encontrar en el valor x, mientras que c_i indica la cantidad de ocurrencias o apariciones de un mismo carácter i dentro del valor 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.

Podemos observar las entropías de algunos valores de ejemplo en la Tabla 2.8. Por ejemplo, comparando los valores *emp* y *empempempempempempem* notamos que tienen la misma cantidad de caracteres distintos y la misma entropía, a pesar de que el segundo tiene mayor longitud.

Eso nos muestra que una mayor longitud del valor no es suficiente para obtener mayores números para la entropía, sino que se necesita también una mayor variación en su contenido. Cabe resaltar que, como se puede observar en los primeros tres valores de la tabla, la presencia de un único carácter resulta en una entropía igual a 0.

Valor	Longitud	Caracteres distintos	Entropía
e	1	1	0,00
ee	2	1	0,00
eeeeeeeeeeeeeeeeeee	20	1	0,00
empleado	8	7	2,75
empleadooooooooooooo	20	7	1,82
emp	3	3	1,58
empempempempempempem	20	3	1,58
empleado@empresa.com	20	12	3,38
abcdefghijklmnopqrst	20	20	4,32

Tabla 2.8: Entropías de valores de ejemplo.

En el trabajo (Nguyen et al., 2011) 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.

Nuestros procesos de extracción de *features* calculan la entropía y retorna ese número para cada valor analizado. En la Tabla 2.9 se puede observar los *features* de entropía extraídos de nuestras peticiones de ejemplo. De vuelta, utilizamos las primeras 10 peticiones para el entrenamiento, mientras que las demás fueron analizadas con los parámetros encontrados en entrenamiento. Cada fila contiene tres componentes del vector $\vec{f_{ij}}$ de la petición correspondiente. Mostramos los números redondeados a dos dígitos de precisión, a pesar de que nuestra implementación utiliza todos los decimales disponibles.

ID de la petición	Petición completa	Parámetro *email*	Parámetro *full*
1	4,88	3,82	0,00
2	4,77	3,61	0,00
3	4,81	3,90	0,00
4	4,94	3,96	0,00
5	4,66	3,46	0,00
6	4,90	3,54	1,58
7	4,85	3,51	0,00
8	4,85	3,94	0,00
9	4,90	3,24	1,58
10	4,81	3,97	0,00
11	4,83	3,54	0,00
12	4,86	3,34	0,00
13	4,91	4,23	0,00
14	4,98	3,88	1,58
15	4,88	3,84	0,00
16	5,36	4,40	0,00
17	4,77	3,70	0,00
18	5,07	3,62	2,32
19	5,03	4,26	0,00
20	4,97	3,48	1,58

Tabla 2.9: Entropía de nuestras 20 peticiones de ejemplo.

2.3.3.3. Cantidad de caracteres

En (Kruegel and Vigna, 2003) se utiliza la longitud de valores como un indicador de anomalías. La longitud puede ser entendida también como la cantidad total de caracteres del valor. En (Nguyen et al., 2011) se 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 a nuestro detector OCS-WAF 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 OCS-WAF detectar la anomalía, lo cual explicaremos en más detalle en el próximo capítulo.

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. La Tabla 2.10 muestra los *features* de cantidad de caracteres extraídos de nuestras peticiones de ejemplo. También para este caso, las primeras 10 peticiones fueron utilizadas para el entrenamiento, mientras que las demás fueron analizadas con los parámetros encontrados en esa primera fase. Cada fila representa 12 componentes del vector de *features* $\vec{f_{ij}}$ de la petición correspondiente.

ID	Petición completa				Parámetro *email*				Parámetro *full*			
	Todos	Díg.	Let.	Otro	Todos	Díg.	Let.	Otro	Todos	Díg.	Let.	Otro
1	88	9	63	16	20	0	18	2	0	0	0	0
2	94	9	69	16	26	0	24	2	0	0	0	0
3	100	9	74	17	32	0	29	3	0	0	0	0
4	97	9	71	17	29	0	26	3	0	0	0	0
5	103	9	78	16	35	0	33	2	0	0	0	0
6	99	10	71	18	22	1	19	2	3	0	3	0
7	83	11	56	16	15	2	11	2	0	0	0	0
8	105	9	78	18	37	0	33	4	0	0	0	0
9	95	9	68	18	18	0	16	2	3	0	3	0
10	99	9	73	17	31	0	28	3	0	0	0	0
11	95	11	68	16	27	2	23	2	0	0	0	0
12	85	9	60	16	17	0	15	2	0	0	0	0
13	108	9	81	18	40	0	36	4	0	0	0	0
14	98	10	70	18	21	1	18	2	3	0	3	0
15	93	9	67	17	25	0	22	3	0	0	0	0
16	130	15	84	31	54	0	37	17	0	0	0	0
17	97	11	72	14	31	4	27	0	0	0	0	0
18	104	17	66	21	17	3	11	3	5	5	0	0
19	132	21	78	33	46	0	27	19	0	0	0	0
20	103	16	66	21	20	1	14	5	3	0	3	0

Tabla 2.10: Cantidad de caracteres de nuestras 20 peticiones de ejemplo.

2.3.4. Composición del vector de *features*

Los vectores de *features* $\vec{f_{ij}}$ representan a cada una de las peticiones HTTP dentro de un grupo G_i. La dimensión de estos vectores está dada por n_i según la Ecuación 2.1. Como ya hemos mencionado, utilizamos un valor de $m = 10$ en nuestros procesos de extracción de *features*. Además, realizando el entrenamiento con las primeras 10 de nuestras peticiones de ejemplo de la Tabla 2.2, obtenemos $|Q_i| = 2$ y $|B_i| = 0$. Con estos números obtenemos $n_i = 30$ y cada petición de ejemplo quedará representada por vectores de 30 componentes. En las secciones anteriores ya fueron presentados estos componentes, específicamente 15 componentes de distribución de caracteres en la Tabla 2.7, luego 3 componentes de entropía en la Tabla 2.9 y finalmente 12 componentes de cantidad de caracteres en la Tabla 2.10. Los vectores finales se obtienen agregando los datos de las tres tablas 2.7, 2.9 y 2.10.

Durante la fase de entrenamiento, nuestros procesos de extracción de *features* trabajan con una cantidad de peticiones recolectadas en cada grupo. Esa cantidad de peticiones en G_i puede ser expresada también como $|G_i|$, con una i distinta para cada grupo de método HTTP y URL.

En el contexto de nuestro ejemplo, para el entrenamiento las primeras 10 peticiones de ejemplo conforman el conjunto G_i correspondiente, resultando en $|G_i| = 10$. El conjunto F_i, que contiene los vectores de *features* de las peticiones del grupo G_i, puede ser expresado también como una matriz numérica M_i, en la cual las filas son los vectores $\vec{f_{ij}}$.

En la Figura 2.2 se puede observar una matriz M_i, que 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 de eso, 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 2.2: 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 posteriormente para entrenar el clasificador One-Class SVM del grupo, lo cual explicaremos en más detalle en el próximo capítulo. 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.

2.3.5. Escalamiento de los *features* extraídos

Los *features* que extraemos de las peticiones HTTP tienen rangos distintos, como se puede observar en la Tabla 2.3. 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 (Rieck, 2009).

De esta manera, nosotros aplicamos un proceso de escalamiento (también denominado normalización por algunos autores) a los datos en las matrices M_i antes de pasarlas a los clasifica-

ID	Distribución de caracteres					Entropía	Cantidad de caracteres			
							Todos	Díg.	Let.	Otro
1	−0,97	0,05	−0,22	0,30	0,56	0,53	−1,31	−0,47	−1,12	−1,08
2	−0,48	1,59	1,15	0,36	−0,92	−0,94	−0,36	−0,47	−0,17	−1,08
3	−0,04	0,50	−0,28	0,42	−0,23	−0,34	0,58	−0,47	0,61	0,12
4	0,15	−0,23	−0,95	−1,64	0,98	1,31	0,11	−0,47	0,14	0,12
5	2,82	1,18	1,63	−1,47	−2,33	−2,32	1,06	−0,47	1,24	−1,08
6	0,02	−0,56	−0,05	−0,70	0,50	0,83	0,43	1,09	0,14	1,32
7	−0,65	−0,41	−2,04	−0,03	1,36	0,20	−2,10	2,65	−2,22	−1,08
8	−0,34	0,85	−0,09	0,67	−0,35	0,16	1,37	−0,47	1,24	1,32
9	−0,54	−1,17	0,90	0,14	0,43	0,87	−0,21	−0,47	−0,33	1,32
10	0,02	−1,79	−0,05	1,95	0,00	−0,31	0,43	−0,47	0,46	0,12
11	0,29	1,40	−1,88	−2,62	0,95	−0,10	−0,21	2,65	−0,33	−1,08
12	−0,78	0,64	−2,53	−0,51	1,38	0,33	−1,78	−0,47	−1,59	−1,08
13	0,22	−1,89	−1,94	1,29	0,88	0,90	1,85	−0,47	1,71	1,32
14	0,09	−1,64	−2,51	−1,83	2,26	1,80	0,27	1,09	−0,02	1,32
15	−0,41	−0,84	−1,43	−0,83	1,46	0,50	−0,52	−0,47	−0,49	0,12
16	0,32	−4,38	−6,47	−4,43	5,71	6,90	5,31	8,90	2,18	16,97
17	−0,66	1,02	−0,95	5,12	−1,59	−0,86	0,11	2,65	0,30	−3,49
18	−1,04	−2,50	−2,41	−0,39	2,74	3,10	1,21	12,03	−0,64	4,94
19	−0,38	−0,87	−4,72	−0,70	2,64	2,57	5,63	18,27	1,24	19,38
20	−1,75	−2,37	−3,49	1,08	2,98	1,67	1,06	10,46	−0,64	4,94

Tabla 2.11: *Features* con escalamiento de la petición completa de nuestras 20 peticiones de ejemplo.

dores One-Class SVM. Mediante este escalamiento se busca que cada uno de los *features* de los datos de entrenamiento, es decir, cada una de las columnas 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* extraídos.

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 2.3 a cada uno de los valores de dicha matriz.

$$x_{\text{nuevo}} = \frac{x_{\text{actual}} - \mu}{\sigma} \tag{2.3}$$

Como se puede observar en la Ecuación 2.3, 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.

ID	Distribución de caracteres					Entropía	Cantidad de caracteres			
							Todos	Díg.	Let.	Otro
1	−1,35	−0,81	0,85	0,07	0,91	0,52	−0,91	−0,47	−0,81	−0,75
2	−0,13	0,08	1,58	−0,15	−0,48	−0,33	−0,07	−0,47	0,04	−0,75
3	−0,07	−0,27	−0,35	−1,19	0,91	0,85	0,77	−0,47	0,75	0,75
4	−0,49	−0,61	−0,80	−0,72	1,29	1,09	0,35	−0,47	0,33	0,75
5	2,22	0,02	0,03	−1,57	−0,64	−0,97	1,19	−0,47	1,32	−0,75
6	−0,52	1,30	1,71	−0,45	−0,74	−0,64	−0,63	1,09	−0,66	−0,75
7	−0,59	1,12	−1,06	1,98	−0,78	−0,77	−1,62	2,65	−1,80	−0,75
8	−0,55	−0,34	−0,44	0,53	0,42	0,99	1,48	−0,47	1,32	2,24
9	1,43	1,44	−0,21	0,71	−1,91	−1,87	−1,19	−0,47	−1,09	−0,75
10	0,04	−1,93	−1,31	0,81	1,03	1,14	0,63	−0,47	0,61	0,75
11	−0,25	1,98	1,20	−1,42	−0,57	−0,63	0,07	2,65	−0,10	−0,75
12	0,39	0,21	2,53	1,08	−1,86	−1,46	−1,33	−0,47	−1,23	−0,75
13	−0,78	−1,54	−1,06	0,07	1,64	2,17	1,90	−0,47	1,74	2,24
14	−0,38	−1,09	−1,43	−0,21	1,43	0,76	−0,77	1,09	−0,81	−0,75
15	0,01	−1,97	0,47	0,07	0,77	0,61	−0,21	−0,47	−0,24	0,75
16	1,01	−1,78	−3,75	−1,95	2,66	2,89	3,86	−0,47	1,88	21,62
17	−0,69	0,87	−0,08	0,81	−0,38	0,04	0,63	5,78	0,47	−3,73
18	−0,95	0,21	0,29	1,08	−0,16	−0,32	−1,33	4,22	−1,80	0,75
19	−1,15	−0,94	−1,23	−0,68	2,01	2,32	2,74	−0,47	0,47	24,60
20	0,92	0,64	−1,06	0,07	−0,54	−0,86	−0,91	1,09	−1,37	3,73

Tabla 2.12: *Features* con escalamiento del parámetro *email* de nuestras 20 peticiones de ejemplo.

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 esta fase de detección, 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.

A continuación, aplicamos este proceso de escalamiento a nuestras peticiones de ejemplo para ilustrar el resultado de este proceso. Los *features* extraídos que presentamos en las tablas 2.7, 2.9 y 2.10 están agrupados según pertenezcan a la distribución de caracteres, entropía o cantidad de caracteres respectivamente. Para una mejor visualización, en esta sección agrupamos los *features* escalados según sean de la petición completa, el parámetro *email* o el parámetro *full*. Estos *features* escalados se pueden ver en las tres tablas 2.11, 2.12 y 2.13 respectivamente.

Cabe volver a mencionar que realizamos el entrenamiento con las primeras 10 peticiones de ejemplo solamente. De esta forma, la matriz M_i correspondiente tendrá 10 filas y 30 columnas, y sobre estos datos se calculan el promedio y la desviación estándar de cada una de las columnas. Posteriormente, para la fase de detección, a las demás peticiones se aplica el mismo cálculo con los promedios y desviaciones obtenidos en el entrenamiento.

ID	Distribución de caracteres					Entropía	Cantidad de caracteres			
							Todos	Díg.	Let.	Otro
1	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
2	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
3	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
4	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
5	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
6	2,00	2,00	0,00	0,00	0,00	2,00	2,00	0,00	2,00	0,00
7	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
8	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
9	2,00	2,00	0,00	0,00	0,00	2,00	2,00	0,00	2,00	0,00
10	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
11	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
12	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
13	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
14	2,00	2,00	0,00	0,00	0,00	2,00	2,00	0,00	2,00	0,00
15	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
16	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
17	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
18	1,00	1,00	0,40	0,00	0,00	3,16	3,67	5,00	−0,50	0,00
19	−0,50	−0,50	0,00	0,00	0,00	−0,50	−0,50	0,00	−0,50	0,00
20	2,00	2,00	0,00	0,00	0,00	2,00	2,00	0,00	2,00	0,00

Tabla 2.13: *Features* con escalamiento del parámetro *full* de nuestras 20 peticiones de ejemplo.

De esta forma, concluimos la explicación de nuestros procesos de extracción de *features* para mensajes HTTP. En el siguiente capítulo presentamos y explicamos el funcionamiento del clasificador One-Class SVM que utilizamos en nuestro detector OCS-WAF para el proceso de detección de mensajes anómalos.

Capítulo 3

Clasificación con One-Class SVM

La detección de anomalías en peticiones HTTP puede ser encarada como un problema de clasificación. Nosotros empleamos la estrategia OCC (Khan and Madden, 2009) para la detección de anomalías en nuestro detector OCS-WAF. Para este fin utilizamos el clasificador One-Class SVM, que es una herramienta del área de ML.

Iniciaremos este capítulo explicando algunos conceptos sobre problemas de clasificación y las herramientas utilizadas para resolverlos, luego describimos el funcionamiento del One-Class SVM y mostramos cómo este clasificador es utilizado para la detección de peticiones anómalas dentro de nuestro WAF.

3.1. Conceptos sobre problemas de clasificación

La detección de anomalías puede ser encarada como un problema de clasificación. En este tipo de problemas se busca clasificar las muestras en varios grupos o clases. Acá 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 la fase de detección. En cambio, se habla de aprendizaje no supervisado cuando no se provee muestras con clases conocidas de antemano y la herramienta utilizada debe tratar de encontrar las clases presentes en las muestras (Torrano-Giménez, 2015). 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 (Aggarwal, 2013).

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.

Una de las estrategias utilizadas para abordar estos dos desafíos es la clasificación de una sola clase (OCC - *One-Class Classification*), que corresponde al tipo semi-supervisado (Khan and Madden, 2009). 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. Esta característica permite que el clasificador no necesite ser entrenado de vuelta con la aparición de novedosas muestras que no pertenecen a la clase conocida. Esta estrategia ha sido utilizada con éxito en varias áreas, como por ejemplo en la detección de spam, reconocimiento de rostros, detección de fallas en maquinarias, entre otros (Khan and Madden, 2014).

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 o ataques son las muestras positivas, que no pertenecerán a la clase conocida de los mensajes normales (que son las muestras negativas).

3.2. Herramientas del área de *Machine Learning*

Para realizar la detección de anomalías en el contexto de problemas de clasificación se puede utilizar varias estrategias y herramientas. Una opción es emplear herramientas estadísticas, como podemos ver en los trabajos (Kruegel and Vigna, 2003), (Giménez, 2015) y (Torrano-Giménez, 2015). Otra opción son herramientas del área de aprendizaje de máquinas (ML - *Machine Learning*) para tratar de detectar las anomalías; podemos ver esto en los trabajos (Sommer and Paxson, 2010), (Buczak and Guven, 2016), (Parhizkar and Abadi, 2015) y (Torrano-Giménez, 2015).

En este trabajo nosotros elegimos la opción de utilizar herramientas del área de ML para tratar de detectar las anomalías. Este tipo de herramientas han sido empleadas con mucho éxito en varias áreas de la computación, como por ejemplo en sistemas de recomendación de productos, clasificación de imágenes, detección de rostros, reconocimiento óptico de caracteres, detección de anomalías, entre otros (Torrano-Giménez, 2015). Podemos ver ejemplos de estos en los trabajos (Sommer and Paxson, 2010), (Buczak and Guven, 2016), (Parhizkar and Abadi, 2015) y (Torrano-Giménez, 2015).

Los algoritmos o herramientas utilizados en ML son muy diversos, como por ejemplo árboles de decisiones (Torrano-Giménez, 2015), redes neuronales (Corchado and Herrero, 2011), algoritmos genéticos (Abadeh et al., 2011), entre otros (Torrano-Giménez, 2015). Una de estas herramientas, que ha sido utilizada con mucho éxito en las tareas de clasificación, es la máquina de vectores de soporte (SVM - *Support Vector Machine*). Una versión modificada del SVM ha sido propuesta como una de varias alternativas para afrontar tareas de OCC (Schölkopf et al., 2001). Varios investigadores ya han empleado exitosamente este clasificador One-Class SVM en problemas de distintas áreas, como por ejemplo en clasificación de textos, clasificación de rostros en imágenes, detección de spam, detección de fallas en máquinas, detección de anomalías, entre otros (Khan and Madden, 2014).

3.3. Funcionamiento del One-Class SVM

Los SVMs son clasificadores binarios de aprendizaje supervisado. Estos clasificadores tratan de separar de forma óptima las dos clases utilizando un hiperplano, buscando minimizar la penalización incurrida por la clasificación incorrecta de alguna de las muestras de entrenamiento. El nombre se debe a que habrá un subconjunto de muestras que determinarán la posición del hiperplano separador; estas muestras son los vectores de soporte que determinan la separación de las clases (Aggarwal, 2013).

Una versión modificada del SVM ha sido propuesta como una de varias alternativas para afrontar tareas de OCC (Schölkopf et al., 2001). Este clasificador One-Class SVM solamente recibe muestras de la clase conocida en la fase de entrenamiento y trata de trazar un hiperplano que separe las muestras del origen de coordenadas. Se busca maximizar la distancia del hiperplano al origen, tratando de minimizar al mismo tiempo la cantidad de muestras situadas en el mismo lado del hiperplano que el origen. Las muestras del lado opuesto al origen serán consideradas pertenecientes a la clase conocida, las demás no pertenecerán a dicha clase (Khan and Madden, 2009).

A continuación presentamos las ecuaciones que definen el comportamiento del clasificador One-Class SVM. Para eso, vamos a utilizar la notación de detección de anomalías con peticiones HTTP que presentamos en el capítulo anterior, a modo de que las ecuaciones se ajusten a la nomenclatura empleada en este trabajo.

3.3.1. Fase de entrenamiento

En esta fase, el clasificador One-Class SVM traza un hiperplano que separa las muestras de entrenamiento del origen. En nuestro contexto de detección de anomalías, nuestro WAF realiza un preprocesamiento de datos en el cual las peticiones HTTP son representadas por medio de vectores numéricos de *features* $\vec{f_{ij}}$. Como explicamos en el capítulo anterior, las peticiones disponibles para el entrenamiento son agrupadas por método y URL, y de cada uno de esos grupos G_i se construye una matriz M_i. Esa matriz tendrá una cantidad de filas igual a la cantidad de peticiones en G_i y una cantidad de columnas igual a la dimensión de los vectores $\vec{f_{ij}}$ del grupo. Después se entrena un clasificador por grupo, usando la matriz M_i para dicho entrenamiento.

3.3.1.1. Formulación del problema de optimización

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 como lo muestra la Ecuación 3.1 (Parhizkar and Abadi, 2015).

$$\vec{w_i} \cdot \vec{x} \; - \; \rho_i \; = \; 0 \tag{3.1}$$

En esta ecuación, $\vec{w_i}$ 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.

Para obtener el hiperplano óptimo, se busca resolver el problema de optimización de la Ecuación 3.2, sujeto a las restricciones de la Ecuación 3.3 (Parhizkar and Abadi, 2015).

$$\min_{\vec{w_i} \, , \, \rho_i \, , \, \xi_i} \; \frac{1}{2}\|\vec{w_i}\|^2 \; - \; \rho_i \; + \; \frac{1}{\nu_i |G_i|} \sum_{j=1}^{|G_i|} \xi_{ij} \tag{3.2}$$

$$\vec{w_i} \cdot \vec{f_{ij}} \geqslant \rho_i - \xi_{ij} \; , \quad \xi_{ij} \geqslant 0 \; , \quad \forall j = 1, 2, \ldots, |G_i| \tag{3.3}$$

Podemos notar dos parámetros fijos en la tarea de minimización de la Ecuación 3.2; $|G_i|$ indica la cantidad de muestras utilizadas para el entrenamiento del clasificador y ν_i es un

parámetro que regula la fracción de muestras situadas al mismo lado del hiperplano que el origen para el grupo G_i. Esas muestras serán clasificadas como no pertenecientes a la clase conocida y así, en el contexto de nuestro WAF, este parámetro puede ser considerado como la fracción aceptable de falsos positivos para el entrenamiento (peticiones HTTP normales detectadas equivocadamente como anomalías). Por otro lado, el parámetro ν_i puede ser de utilidad en caso de que haya algunas muestras no pertenecientes a la clase conocida en los datos de entrenamiento. Para esos casos, el clasificador puede dejar ciertas muestras al mismo lado del hiperplano que el origen, obteniendo de esta forma cierta robustez frente a la presencia de ruido en los datos de entrenamiento (Parhizkar and Abadi, 2015).

La tarea de minimización tiene tres variables para las cuales el clasificador busca la combinación óptima; el vector $\vec{w_i}$ que define el hiperplano del grupo, la distancia ρ_i del hiperplano al origen y en tercer lugar la lista de valores de holgura ξ_i, que indica las penalizaciones incurridas por muestras situadas al mismo lado del hiperplano que el origen (estas muestras serán falsos positivos) (Parhizkar and Abadi, 2015).

Cada una de estas variables domina uno de los tres términos del problema de minimización. El primer término esta gobernado por la norma l_2 del vector $\vec{w_i}$, causando que el clasificador busque el menor valor posible para dicha norma. El segundo término consta de la distancia del hiperplano al origen pero tomado con signo negativo, lo que provoca que el clasificador busque maximizar dicha distancia. El último término indica la penalización total incurrida por las muestras de entrenamiento situadas al mismo lado del hiperplano que el origen (estas muestras serán falsos positivos), causando que el clasificador trate de minimizar dichos casos (Parhizkar and Abadi, 2015).

En las restricciones presentadas en la Ecuación 3.3 podemos observar que cada una de las muestras (en nuestro caso, los vectores $\vec{f_{ij}}$) debe estar ubicada a una distancia mayor o igual a $\rho_i - \xi_{ij}$ del origen. En el caso ideal, los valores de holgura ξ_{ij} para todas las muestras son 0, lo que significa que ninguna de las muestras se encuentra del mismo lado del hiperplano que el origen. En cambio, si alguna muestra de entrenamiento se encuentra en el mismo lado que el origen (será un falso positivo), la holgura correspondiente a esa muestra deberá tener un valor mayor a 0 para satisfacer las restricciones, provocando que el problema de minimización tendrá una penalización que debe considerar. Esto otorga cierta flexibilidad al clasificador para balancear entre el aumento de la distancia ρ_i (alejando el hiperplano del origen) o la reducción de la cantidad de elementos en ξ_i con valores mayores a 0 (reduciendo la cantidad de falsos positivos) (Parhizkar and Abadi, 2015).

3.3.1.2. Transformación a espacios de dimensiones mayores

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 (Parhizkar and Abadi, 2015). Esta transformación puede ser expresada como funciones $\phi : \mathbb{R}^n \to \mathbb{R}^m$, 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 $\mathbb{R}^n$.

Para obtener esta transformación, se utiliza funciones ϕ para modificar los vectores de muestras en las ecuaciones 3.1 y 3.3, obteniendo de esta forma las nuevas ecuaciones 3.4 y 3.5 (Parhizkar and Abadi, 2015) (Amer et al., 2013). En estas ecuaciones, el vector $\vec{w_i}$ del hiperplano se encuentra también en el nuevo espacio $\mathbb{R}^m$. Cabe aclarar que la formulación del problema de minimización que presentamos en la Ecuación 3.2 no sufre modificaciones por la introducción de esta transformación.

$$\vec{w_i} \cdot \phi(\vec{x}) \ - \ \rho_i \ = \ 0 \tag{3.4}$$

$$\vec{w_i} \cdot \phi(\vec{f_{ij}}) \geqslant \rho_i - \xi_{ij} \ , \quad \xi_{ij} \geqslant 0 \ , \quad \forall j = 1, 2, \ldots, |G_i| \tag{3.5}$$

En la Figura 3.1 se puede observar un ejemplo sencillo que ilustra como la transformación de las muestras a otro espacio vectorial puede ayudar a trazar el hiperplano. Los círculos rojos indican muestras de la clase conocida utilizadas en entrenamiento, mientras que los cuadrados azules representan las muestras analizadas durante la fase de detección que no pertenecen a la clase conocida. Cabe aclarar que durante la fase de entrenamiento el clasificador no tiene disponible los cuadrados azules para buscar el hiperplano óptimo. Se puede notar que el clasificador logra trazar un hiperplano en el nuevo espacio $\mathbb{R}^m$ para separar los círculos rojos del origen, y así finalmente también de los cuadrados azules. Ese hiperplano puede tener una representación no lineal en el espacio original $\mathbb{R}^n$, como se puede ver también en la figura.

3.3.1.3. Funciones *kernel*

La transformación de muestras a espacios de mayor dimensión trae consigo la dificultad de realizar las multiplicaciones de vectores en dichos espacios. Las multiplicaciones presentes en

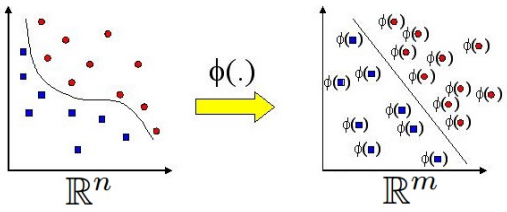

Figura 3.1: Diagrama de transformación de muestras del espacio vectorial $\mathbb{R}^n$ a un nuevo espacio $\mathbb{R}^m$, con $n \leq m$.

las ecuaciones 3.4 y 3.5 pueden 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 (Rieck, 2009).

Para superar este desafío, se puede emplear unas funciones denominadas *kernels*, que permiten omitir esas multiplicaciones de vectores en el nuevo espacio. Los *kernels* comparan dos vectores del espacio original $\mathbb{R}^n$ y retornan un valor escalar que indica la similitud que dichos vectores presentan en el nuevo espacio $\mathbb{R}^m$ de mayor dimensión. El resultado producido por estas funciones equivale al producto escalar de los dos vectores en el nuevo espacio $\mathbb{R}^m$, como se puede observar también en la Ecuación 3.6, pero con la ventaja de que no se realiza realmente esta multiplicación costosa (Rieck, 2009).

$$K(\vec{x_1}, \vec{x_2}) \;=\; \phi(\vec{x_1}) \cdot \phi(\vec{x_2}) \tag{3.6}$$

Estos *kernels* no son exclusivos de los clasificadores One-Class SVM ni de SVMs en general, pero se ha utilizado estas funciones con mucho éxito en este tipo de clasificadores. Existen varias opciones de kernels que pueden utilizarse, los más conocidos son el *kernel* lineal, el polinomial, el sigmoidal y el *Radial Basis Function* (RBF) *kernel* (Rieck, 2009). Usando el *kernel* lineal, el hiperplano separador queda representado por un hiperplano también en el espacio original; los otros tres *kernels* mencionados generan superficies no lineales en el espacio original. Realizando varias pruebas para comparar la efectividad de estos *kernels*, el RBF es el que arrojó mejores resultados. Además, varios trabajos relacionados han empleado este *kernel* con éxito para sus tareas de clasificación, como podemos ver en los trabajos (Parhizkar and Abadi, 2015) y (Perdisci et al., 2006). Por estos motivos, nosotros utilizamos el RBF en los clasificadores dentro de nuestro detector OCS-WAF. La formulación del *kernel* RBF puede ser expresada según la Ecuación 3.7 (Parhizkar and Abadi, 2015).

$$K(\vec{x_1}, \vec{x_2}) \;=\; \phi(\vec{x_1}) \cdot \phi(\vec{x_2}) \;=\; \exp(-\gamma \|\vec{x_1} - \vec{x_2}\|^2) \tag{3.7}$$

En esta ecuación, se puede ver que el *kernel* obtiene el cuadrado de la norma l_2 (o distancia euclidiana) de la diferencia de dos vectores en el espacio original. Este resultado escalar es multiplicado por un factor γ, usando este nuevo resultado en una exponenciación con la base de los logaritmos naturales. Estas operaciones son computacionalmente más eficientes que la multiplicación de vectores en espacios de mayores dimensiones (Rieck, 2009).

En el contexto del One-Class SVM, el parámetro γ_i determina la dimensión de la región de influencia de cada muestra de entrenamiento del grupo G_i. Esto regula la cercanía necesaria de nuevas muestras a las muestras de entrenamiento para ser consideradas pertenecientes a la clase conocida. Este parámetro tiene relación inversa al radio de la región de influencia de las muestras. Esto significa que valores menores de γ_i aumentan el radio de la región de influencia de las muestra de entrenamiento, causando que la superficie separadora que representa al hiperplano en el espacio original $\mathbb{R}^n$ se encuentre más alejada de dichas muestras. En cambio, valores mayores de γ_i reducen este radio de influencia, permitiendo una superficie más cercana a las muestras vistas.

La Figura 3.2 muestra, con un ejemplo de peticiones HTTP, la superficie separadora no lineal obtenida en el espacio original con dos *kernels*. El RBF permite detectar más anomalías (círculos amarillos) que el *kernel* lineal, que solamente puede trazar una superficie lineal en el espacio original. Además, se ilustra la influencia del parámetro γ_i en la forma de la superficie separadora, mostrando que γ_i mayor genera una superficie más ajustada a las muestras de entrenamiento.

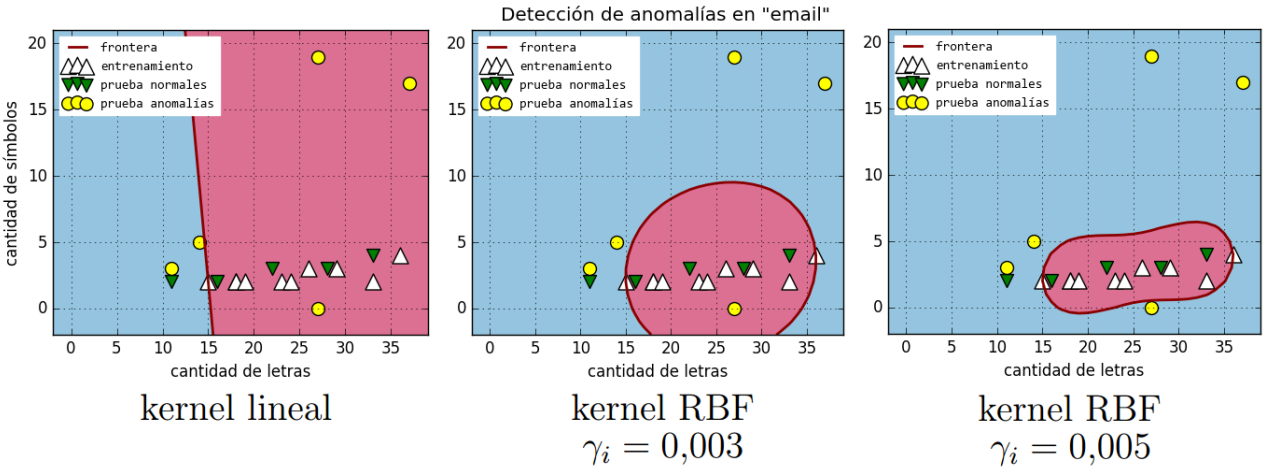

Figura 3.2: Comparación de superficies separadoras obtenidas por el One-Class SVM con distintos *kernels* y parámetros.

3.3.1.4. Formulación dual de la optimización

Para aprovechar la facilidad de cálculo que brindan los *kernels*, es conveniente que el clasificador One-Class SVM no resuelva directamente el problema de optimización de la Ecuación 3.2, sino que resuelva la formulación dual del problema. Esta formulación dual queda expresada como se puede observar en la Ecuación 3.8, sujeto a las restricciones que muestra la Ecuación 3.9 (Aggarwal, 2013).

$$\min_{a_i} \ \frac{1}{2} \ a_i \ S_i \ a_i^T \tag{3.8}$$

$$\sum_{j=1}^{|G_i|} a_{ij} \ = \ 1 \ , \quad 0 \leqslant a_{ij} \leqslant \frac{1}{\nu_i |G_i|} \ \forall j = 1, 2, \ldots, |G_i| \tag{3.9}$$

En estas ecuaciones, S_i es una matriz simétrica que representa los productos escalares de todas las muestras de entrenamiento multiplicadas entre ellas en el espacio de dimensiones mayores $\mathbb{R}^m$; ambas dimensiones de esta matriz equivalen a la cantidad de muestras $|G_i|$. El vector a_i contiene una cantidad $|G_i|$ de coeficientes variables que corresponden a cada una de las muestras, y la formulación dual de la optimización busca encontrar estos coeficientes. De esta forma, la minimización consta de la multiplicación de un vector fila a_i por una matriz simétrica S_i, y el vector fila resultante es multiplicado por el vector columna a_i^T, lo que resulta finalmente en un valor escalar a ser minimizado.

Las muestras de entrenamiento que tengan coeficiente a_{ij} mayor a 0 serán los vectores de soporte del clasificador. Mayormente, los vectores de soporte son un subconjunto reducido de las muestras de entrenamiento, resultando finalmente en un vector a_i con la mayoría de los coeficientes iguales a 0 (Perdisci et al., 2006).

La matriz simétrica S_i, que está presente en la formulación dual de la optimización, representa las multiplicaciones de vectores que se busca omitir por su alto costo computacional. Esta matriz puede ser reemplazada por los resultados de un *kernel*. Esto se logra construyendo una matriz con las mismas dimensiones que S_i, pero cuyos elementos son obtenidos mediante un *kernel* en vez de multiplicación de vectores.

Después de resolver la minimización de la formulación dual, que presentamos en la Ecuación 3.8, el clasificador One-Class SVM ya tiene disponible el vector de coeficientes a_i del grupo G_i. Entonces, el hiperplano definido en la Ecuación 3.4 también puede ser expresado en términos del *kernel* K_i que fue utilizado para sustituir S_i en la formulación dual y el vector a_i obtenido por la minimización.

De esta forma, el hiperplano del clasificador puede quedar definido como lo muestra la Ecuación 3.10 (Aggarwal, 2013).

$$\vec{w_i} \cdot \phi(\vec{x}) \ - \ \rho_i \ = \ \sum_{j=1}^{|G_i|} \Big(a_{ij} \, K_i(\vec{f_{ij}}, \vec{x}) \Big) \ - \ \rho_i \ = \ 0 \tag{3.10}$$

Como ya mencionamos, utilizamos el *kernel* RBF en nuestro WAF. Reemplazando la fórmula de este *kernel*, que fue presentada en la Ecuación 3.7, podemos definir el hiperplano de la Ecuación 3.10 de la forma como lo expresa la Ecuación 3.11, donde γ_i es el parámetro del *kernel* para el clasificador del grupo G_i.

$$\vec{w_i} \cdot \phi(\vec{x}) \ - \ \rho_i \ = \ \sum_{j=1}^{|G_i|} \Big(a_{ij} \, \exp(-\gamma_i \| \vec{f_{ij}} - \vec{x} \|^2) \Big) \ - \ \rho_i \ = \ 0 \tag{3.11}$$

De esta forma, el clasificador One-Class SVM puede construir un hiperplano separador a través de la simulación de multiplicación de vectores en espacios de altas dimensiones, pero sin incurrir realmente en los elevados costos computacionales que implicaría la realización de esas multiplicaciones.

3.3.2. Fase de detección

Después de la fase de entrenamiento, el clasificador 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 (Perdisci et al., 2006). 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. Luego se analiza la posición de ese vector con respecto al hiperplano trazado por el clasificador del grupo.

En esta fase de detección, el clasificador One-Class SVM emplea la fórmula del hiperplano en una función de decisión con el fin de determinar si una muestra nueva pertenece a la clase conocida o no. En la Ecuación 3.12 se puede observar la formulación de esta función de decisión en términos del hiperplano (Amer et al., 2013).

$$g_i(\vec{x}) \ = \ \begin{cases} \vec{w_i} \cdot \vec{x} - \rho_i \ \geqslant \ 0 & +1 \\ \text{caso contrario} & -1 \end{cases} \tag{3.12}$$

En esta ecuación, $\vec{w_i}$ es el vector que define el hiperplano del grupo, $\vec{x}$ es el vector que representa las nuevas muestras a analizar, ρ_i es la distancia del hiperplano al origen y $g_i(\vec{x})$ es la función de decisión, que recibe el vector $\vec{x}$ como su argumento. De esta forma, se obtendrá $g_i(\vec{x}) = 1$ para las muestras que se encuentran separadas del origen por el hiperplano, indicando que las mismas pertenecen a la clase conocida, pero se tendrá $g_i(\vec{x}) = -1$ para las muestras que se encuentren en el mismo lado que el origen, indicando que estas últimas no pertenecen a dicha clase.

En el contexto de nuestro WAF, nuevas peticiones HTTP serán representadas por vectores de *features* $\vec{x}$, siempre atendiendo que las dimensiones de esos vectores sean correspondientes a sus grupo G_i respectivos. La función $g_i(\vec{x})$ retorna el valor positivo para peticiones detectadas como normales (que serán identificadas como muestras negativas) y devuelve el valor negativo para aquellas peticiones detectadas como anomalías (que serán identificadas como muestras positivas o ataques).

Para vectores en espacios de dimensiones mayores, esta función de decisión incorpora las ya mencionadas funciones de transformación ϕ, y puede ser expresada de la forma que lo indica la Ecuación 3.13 (Amer et al., 2013).

$$g_i(\vec{x}) \;=\; \begin{cases} \vec{w_i} \cdot \phi(\vec{x}) - \rho_i \;\geqslant\; 0 & +1 \\ \text{caso contrario} & -1 \end{cases} \tag{3.13}$$

Empleando las optimizaciones que nos brindan los *kernels*, podemos sustituir la multiplicación de vectores en esta función. De esta forma, la función de decisión que expresada según la Ecuación 3.14 (Perdisci et al., 2006).

$$g_i(\vec{x}) \;=\; \begin{cases} \sum_{j=1}^{|G_i|} \left(a_{ij}\, K_i(\vec{f_{ij}}, \vec{x}) \right) - \rho_i \;\geqslant\; 0 & +1 \\ \text{caso contrario} & -1 \end{cases} \tag{3.14}$$

Finalmente, reemplazando en esta ecuación la fórmula del *kernel* RBF, obtenemos una función según la Ecuación 3.15, donde γ_i es el parámetro del *kernel* para el clasificador del grupo G_i correspondiente.

$$g_i(\vec{x}) \;=\; \begin{cases} \sum_{j=1}^{|G_i|} \left(a_{ij}\, \exp(-\gamma_i \|\vec{f_{ij}} - \vec{x}\|^2) \right) - \rho_i \;\geqslant\; 0 & +1 \\ \text{caso contrario} & -1 \end{cases} \tag{3.15}$$

Se puede observar que la función de decisión, que se utiliza para determinar de que lado del hiperplano cae una nueva muestra, se reduce a calcular la norma l_2 entre la muestra nueva

y los vectores de soporte (aquellos con coeficientes a_{ij} mayor a 0). Considerando que suele haber una cantidad reducida de vectores de soporte en un clasificador entrenado, esta función de decisión $g_i(\vec{x})$ resulta en computaciones rápidas y eficientes (Perdisci et al., 2006). De esta manera mostramos que el clasificador One-Class SVM es una herramienta con un bajo costo computacional durante la fase de detección, y por lo tanto puede ser empleado en nuestro detector OCS-WAF para realizar la detección en tiempo real de mensajes HTTP anómalos.

3.4. Trabajos relacionados con One-Class SVM

En la sección anterior describimos el funcionamiento del clasificador One-Class SVM, mostrando las ecuaciones que definen su comportamiento durante las fases de entrenamiento y detección. A continuación presentamos algunos trabajos relacionados del área de IDS que utilizan este clasificador.

En (Nguyen et al., 2016) se presenta un detector llamado *POCAD*. Este detector extrae conjuntos de *n-grams* de *bytes* de paquetes IP y peticiones HTTP y luego aplica un proceso de *clustering* para reducir la cantidad de *features*; OCS-WAF utiliza los procesos de extracción de *features* presentados en el Capítulo 2, que no incluyen *n-grams*. Posteriormente, *POCAD* utiliza un One-Class SVM para realizar la detección de anomalías, de manera similar a nuestra implementación. Los autores realizan pruebas comparativos con otros sistemas de detección, concluyendo que su implementación logra una mejor detección que los demás.

En (Parhizkar and Abadi, 2015) se presenta un detector de anomalías llamado *OC-WAD* para peticiones HTTP. Los autores extraen una cantidad fija de *features* de las peticiones, a diferencia de nuestros procesos de extracción que producen una cantidad variable de *features* que depende de los parámetros presentes en las peticiones. Posteriormente, *OC-WAD* utiliza un conjunto de One-Class SVM con *kernel* RBF para realizar la detección, poniendo énfasis en la creación y optimización del conjunto de clasificadores mediante un algoritmo de enjambres llamado *BeeSnips*. Nuestro OCS-WAF utiliza también el mismo *kernel* y clasificador, pero se entrena un clasificador por grupo de método y URL. Los autores concluyen mediante algunas comparaciones que su implementación supera algunos otros trabajos que mencionan.

En (Perdisci et al., 2006) presentan un detector que extrae conjuntos de *n-grams* de *bytes* de peticiones HTTP. También se aplica un proceso de *clustering* sobre los *features* extraídos. Como ya explicamos, nuestros procesos de extracción no analizan *n-grams*. Los autores emplean un conjunto de One-Class SVM con el *kernel* RBF; cada uno de los clasificadores es entrenado sobre distintos subconjuntos de *features*. En cambio, OCS-WAF utiliza también el mismo *kernel*

y clasificador, pero se entrena un clasificador por grupo de método y URL. Las conclusiones de los autores subrayan la robustez de su detector frente a ataques que tratan de ocultar su firma utilizando diversas técnicas de ofuscación.

En (Tran et al., 2004) se presenta un detector de anomalías llamado *OTAD* para analizar paquetes IP. Los autores utilizan una cantidad fija de cinco *features* que corresponden a datos obtenidos mediante la herramienta *tcpstat*; nuestra implementación extrae una cantidad variable de *features* de peticiones HTTP. Posteriormente, *OTAD* emplea un clasificador One-Class SVM con el *kernel* RBF, utilizando un algoritmo genético para la optimización de la selección de parámetros. En cambio, OCS-WAF utiliza también el mismo *kernel* y clasificador, pero sin mecanismo para la selección automática de parámetros. Los autores concluyen que su detector obtiene buenos resultados, con la ventaja de poder utilizar directamente los datos del *tcpstat* sin necesidad de conversiones o transformaciones.

En (Rieck, 2009) se propone un detector de anomalías que trabaja sobre un conjunto variado de *features* de peticiones HTTP, incluyendo secuencias y árboles para representar relaciones en los datos. Emplea un tipo de One-Class SVM que esta implementado de forma distinta a lo que presentamos en este capítulo; el autor usa un clasificador que emplea una hiperesfera en vez de un hiperplano para clasificar las muestras. Concluye que el detector presentado detecta una gran variedad de ataques, resaltando la baja tasa de falsos positivos.

El éxito en la detección de anomalías que reportan estos trabajos nos confirma que el clasificador One-Class SVM es una herramienta útil para nuestro WAF.

3.5. Clasificación de peticiones de ejemplo

Después de haber explicado el funcionamiento del clasificador One-Class SVM, a continuación ilustramos el proceso de clasificación y detección de anomalías HTTP que realiza nuestro detector OCS-WAF. En el Capítulo 2 hemos introducido un ejemplo mediante el cual explicamos nuestros procesos de extracción de *features*. En la Tabla 2.2 se pudo observar 20 peticiones de ejemplo, con las cuales realizamos la extracción y el escalamiento de los *features*. Los números finales que representan las peticiones fueron presentados en las tablas 2.11, 2.12 y 2.13.

Para el proceso de clasificación, entrenamos un One-Class SVM con la matriz M_i obtenida a partir de nuestras peticiones de ejemplo, la cual tiene 10 filas y 30 columnas. Para la selección de los valores para ν_i y γ_i realizamos el entrenamiento con varios valores en el rango $[0{,}0001; 0{,}5]$, seleccionando los valores con los mejores resultados en la fase de detección. De esta forma, para este ejemplo utilizamos los valores 0,2778 y 0,0001 para ν_i y γ_i respectivamente.

ID	Grupo	Vector de soporte	Clasificación real	Clasificación obtenida	Correctamente clasificado
1	entrenamiento	-	normal	normal	Sí
2	entrenamiento	-	normal	normal	Sí
3	entrenamiento	-	normal	normal	Sí
4	entrenamiento	-	normal	normal	Sí
5	entrenamiento	Sí	normal	normal	Sí
6	entrenamiento	-	normal	normal	Sí
7	entrenamiento	Sí	normal	normal	Sí
8	entrenamiento	Sí	normal	normal	Sí
9	entrenamiento	Sí	normal	normal	Sí
10	entrenamiento	-	normal	normal	Sí
11	detección	-	normal	normal	Sí
12	detección	-	normal	normal	Sí
13	detección	-	normal	anomalía	-
14	detección	-	normal	anomalía	-
15	detección	-	normal	normal	Sí
16	detección	-	anomalía	anomalía	Sí
17	detección	-	anomalía	anomalía	Sí
18	detección	-	anomalía	anomalía	Sí
19	detección	-	anomalía	anomalía	Sí
20	detección	-	anomalía	anomalía	Sí

Tabla 3.1: Resultados de clasificación de nuestras 20 peticiones de ejemplo.

En la Tabla 3.1 se pueden observar los resultados del entrenamiento y la posterior detección de todas las peticiones de ejemplo. Se puede notar que, como se ha explicado en este capítulo, solamente algunas de las muestras de entrenamiento (que son las primeras 10 peticiones) son utilizados como vectores de soporte del clasificador. Además, la tabla muestra que 18 de las 20 peticiones fueron clasificadas correctamente por el One-Class SVM entrenado; solamente dos peticiones normales fueron categorizadas incorrectamente, mientras que las cinco anomalías del conjunto de ejemplo fueron detectadas por el clasificador.

Como muestran los resultados, la clasificación no siempre es exitosa, ya que depende de las muestras con las que debe trabajar el clasificador. No podemos visualizar en dos o tres dimensiones el espacio original de 30 *features*, ni mucho menos el espacio de dimensiones superiores. De esta forma, no hemos logrado analizar la superficie separadora que representa al hiperplano en el espacio original; el análisis de las componentes principales (PCA) tampoco pudo brindar aportes en este análisis. La idea de este ejemplo es también resaltar esta realidad en las tareas de detección de anomalías, mostrando una situación en la cual no se logra la detección correcta de todas las muestras.

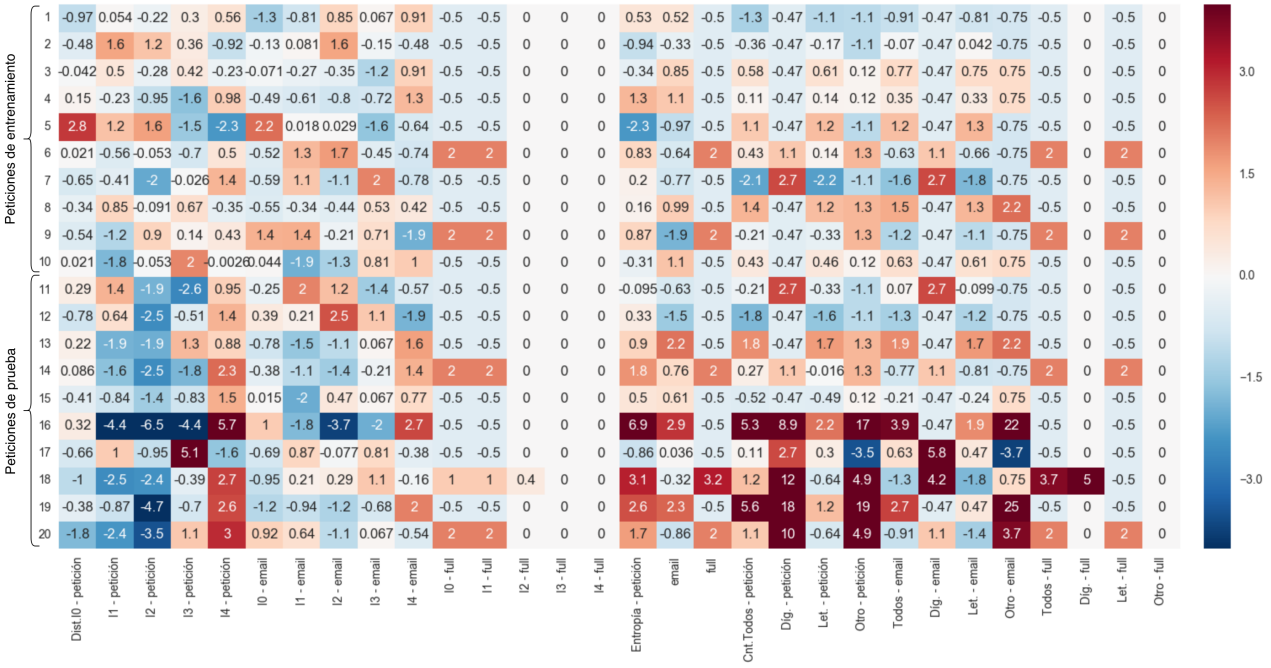

Figura 3.3: Mapa de calor de los *features* extraídos de nuestras 20 peticiones de ejemplo.

En la Figura 3.3 se puede observar los números escalados de todos nuestros *features* para las 20 peticiones de ejemplo, desplegadas dentro de un gráfico del tipo mapa de calor (*heatmap*) para permitir una mejor visualización de los rangos de valores. Cada *features* fue escalado para que su promedio sea 0, que en el gráfico corresponde al color blanco; regiones de colores rojo o azul oscuros indican mayor alejamiento del 0. Se puede notar, por ejemplo, que las peticiones 16 al 20, que contienen anomalías, tienen mayores ocurrencias de colores oscuros que indican que esos *features* están alejados del promedio; esto le permite al One-Class SVM detectar estas anomalías.

En cambio, la petición 14 tiene un valor mucho más elevado para el último intervalo de la distribución de caracteres de la petición completa (quinta columna desde la izquierda) que todas las peticiones de entrenamiento, lo que pudo haber provocado que el clasificador no detectó esa petición como normal. Lo mismo se da para la petición 13, que tiene una entropía del parámetro *email* más elevada que las peticiones de entrenamiento.

Esta representación de los *features* puede ser de utilidad para continuar el análisis de los datos, inspeccionando las relaciones entre los *features* extraídos y los resultados de clasificación mostrados en la Tabla 3.1.

Capítulo 4

Implementación de OCS-WAF

Este capítulo presenta los detalles de implementación de OCS-WAF, un WAF que utiliza los componentes descritos en los capítulos anteriores para la detección de mensajes HTTP anómalos. Iniciaremos este capítulo explicando algunos conceptos sobre WAFs y detección de anomalías, luego presentamos la arquitectura general de nuestra implementación y después describimos el funcionamiento de la misma durante las fases de entrenamiento y detección, detallando los pasos intermedios presentes en cada una de las dos fases.

El código fuente de nuestra implementación está disponible en nuestro repositorio público bajo la dirección `https://github.com/nico-ralf-ii-fpuna/tfg`.

4.1. Detección de intrusiones con *Web Application Firewalls*

Los sistemas de detección de intrusión (IDS - *Intrusion Detection System*) son programas o dispositivos especializados para monitorear las actividades en un sistema en busca de intrusiones no autorizadas o posibles ataques, siendo un elemento fundamental y necesario para garantizar la seguridad del sistema (Scarfone and Mell, 2007).

Se pueden utilizar tres criterios principales para clasificar los IDS, en primer lugar según los modos de respuesta que utilizan frente a intrusiones, en segundo lugar según las fuentes de datos que emplean para sus análisis y en tercer lugar se los puede clasificar según la metodología de detección que usan (Torrano-Giménez, 2015). A continuación damos una breve explicación de los IDS según cada uno de estos criterios de clasificación, resaltando las características más relevantes de cada caso.

- De acuerdo al modo de respuesta del sistema, se puede tener:
  - *Intrusion Detection System* (IDS): esta clase de sistema actúa de forma pasiva y solamente lanza alertas o mensajes cuando detecta actividades no autorizadas, pero no realiza acciones de contención.
  - *Intrusion Prevention System* (IPS): este tipo de sistema trabaja de forma activa para prevenir intrusiones y está equipado con herramientas para mitigar los daños de posibles ataques.
- De acuerdo a las fuentes de datos para el análisis, se puede tener:
  - *Host-based systems* (HIDS): esta clase de sistema analiza las actividades de máquinas individuales, monitoreando diferentes aspectos de las mismas, como por ejemplo las aplicaciones abiertas, los procesos en ejecución, los accesos y las modificaciones de archivos, entre otros.
  - *Network-based systems* (NIDS): este tipo de sistema analiza el tráfico que pasa por las redes de comunicación, monitoreando a nivel de paquetes IP o también a nivel de mensajes HTTP. Normalmente, estos sistemas son colocados en puntos de entrada a una red, o frente a sistemas críticos, como por ejemplo servidores. En el caso de que se analicen mensajes HTTP se puede hablar específicamente de cortafuegos para aplicaciones web (WAF - *Web Application Firewall*).
- De acuerdo a la metodología de detección, se puede tener:
  - *Signature-based detection*: esta clase de sistemas busca patrones de ataques a partir de una lista de firmas de ataques conocidos.
  - *Anomaly-based detection*: este tipo de sistemas busca desviaciones del comportamiento normal o anomalías en las fuentes de datos que monitorea, ya que estas anomalías pueden indicar intrusiones o ataques.
  - *Hybrid systems*: se puede combinar también los sistemas de detección por firmas y anomalías, para tratar de aprovechar las ventajas que provee cada uno.

Considerando estos conceptos expuestos, según el modo de respuesta que pueden tener los sistemas, nuestra implementación es un IPS por poseer mecanismos para bloquear mensajes anómalos; esos bloqueos pueden ser desactivadas para utilizar solamente detección pasiva. Por

otro lado, como nuestra implementación utiliza mensajes HTTP como su fuente de datos para el análisis, OCS-WAF puede ser considerado un sistema NIDS.

En tercer lugar, considerando la metodología de detección, OCS-WAF utiliza *anomaly-based detection*, ya que este método tiene ventajas sobre la detección por firmas. Para que un WAF pueda utilizar eficazmente el método por firmas, es necesario que el mismo mantenga una lista actualizada de las firmas de los ataques conocidos. La lista de firmas de ataques descubiertos crece constantemente y probablemente nunca deje de crecer. Durante el análisis de los 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 para este proceso de detección (Kruegel and Vigna, 2003).

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. Se basa en la premisa de que los ataques se diferencian en alguna forma de los mensajes 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, para detectar aquellos mensajes HTTP que son considerados anomalías con respecto a los mensajes normales vistos durante entrenamiento (Kruegel and Vigna, 2003).

Para WAFs basados en anomalías, la fase de entrenamiento es obligatoria una vez al inicio del uso y después solamente si existen cambios en los mensajes normales, por ejemplo después de una modificación a una de las aplicaciones web protegidas por el WAF en cuestión. El método de detección por anomalías tiene la ventaja de poder detectar anomalías debidas a nuevos tipos de ataques desde el momento que aparezcan, mientras que los métodos por firmas dependen de la actualización de su lista de ataques conocidos (Kruegel and Vigna, 2003).

4.2. 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. La arquitectura general de OCS-WAF se puede observar en la Figura 4.1.

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. En las siguientes secciones de este capítulo presentamos los pasos intermedios que componen cada fase.

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 (Lagadec, 2011). 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, lo que se puede observar también en la Figura 4.1, pero nuestra implementación puede ser extendida para incluir el análisis de las respuestas que son retornadas por las aplicaciones protegidas. Dentro de nuestro repositorio, la base modificada en cuestión se encuentra en los archivos `/waf/proxy_base.py` y también `/waf/proxy_implementation.py`, mientras que nuestros procesos de extracción de *features* están en los archivos del directorio `/waf/feature_extraction/`.

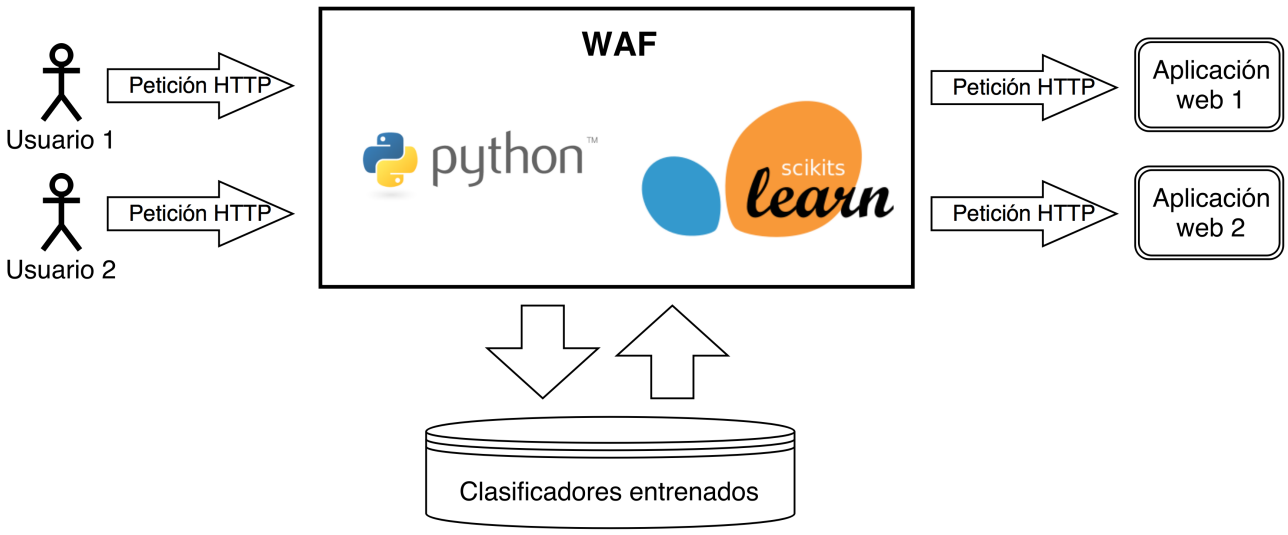

Figura 4.1: Arquitectura general del funcionamiento de OCS-WAF.

OCS-WAF también utiliza la librería *scikit-learn* (Pedregosa et al., 2011), 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.

4.3. Fase de entrenamiento

Esta fase sirve para preparar el OCS-WAF para la detección de peticiones HTTP anómalas. La Figura 4.2 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.

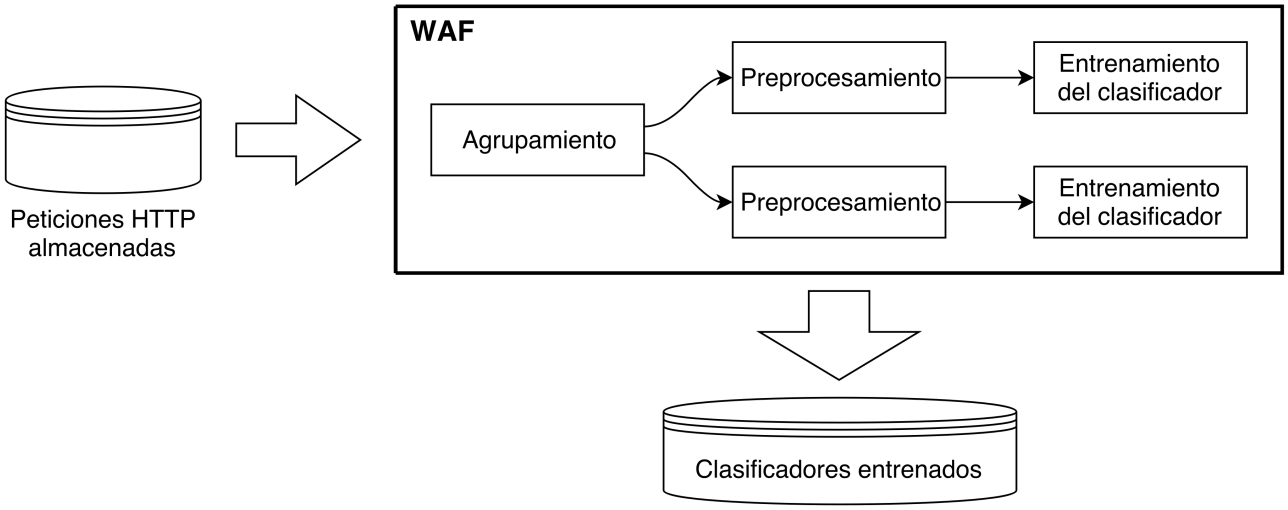

Figura 4.2: Arquitectura de OCS-WAF en la fase de entrenamiento.

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; para eso se debe proceder a deshabilitar los procesos de análisis y solamente almacenar las peticiones que pasen por el detector.

4.3.1. Paso de agrupamiento

Como ya hemos explicado en el Capítulo 2, 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.

Utilizando la nomenclatura que hemos introducido en el Capítulo 2, podemos denominar G al conjunto de los grupos de peticiones que se obtienen por este paso de agrupamiento. Consecuentemente, cada grupo es identificado como G_i.

4.3.2. 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 4.2 podemos ver múltiples instancias de preprocesamiento, que corresponden a los distintos grupos.

Primeramente, como ya hemos explicado en el Capítulo 2, 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. Las funciones utilizadas para la extracción de estos *features* se encuentran en el directorio `/waf/feature_extraction/` de nuestro repositorio. Debido a que los *features* ya fueron explicados en detalle en el Capítulo 2, a continuación nos limitaremos a una breve descripción de las tres funciones utilizadas por OCS-WAF para obtener los *features* para cada valor analizado.

- *Distribución de caracteres*: se calcula la frecuencia relativa de cada carácter, se ordena estas frecuencias de forma descendente y se agrupan las mismas en cinco intervalos de distintos tamaños. Para cada valor analizado, esta función retorna cinco números del tipo punto flotante, que corresponden a los cinco intervalos que fueron calculados.

- *Entropía*: está función calcula la entropía de un valor según la Ecuación 2.2. Para cada valor analizado, se retorna un número de tipo punto flotante, que corresponde a la entropía calculada.

- *Cantidad de caracteres*: esta función cuenta los caracteres de un valor que pertenecen a cuatro categorías definidas, específicamente contando la cantidad total de caracteres, la cantidad de dígitos, de letras, y finalmente la cantidad de otros caracteres que no sean dígitos ni letras. Para cada valor analizado, la función retorna cuatro números de tipo punto flotante, que corresponden a las cantidad de las cuatro categorías mencionadas. Esta función podría retornar números enteros, pero elegimos el tipo de dato punto flotante para que se obtenga vectores $\vec{f_{ij}}$ homogéneos sin realizar conversiones adicionales posteriormente.

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 2.1. 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.

4.3.3. 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. Como ya mencionamos en el Capítulo 3, 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. Esto se debe a que durante la fase de entrenamiento no se cuenta con peticiones anómalas (muestras positivas) para validar si el clasificador entrenado clasifica correctamente las anomalías; solamente se puede validarlo contra las peticiones normales. Esto es un desafío que está presente en los problemas de OCC, ya que no se tiene disponible mucho conocimiento sobre las muestras que no pertenecen a la clase conocida (Khan and Madden, 2009).

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.

4.4. 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 4.3 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.

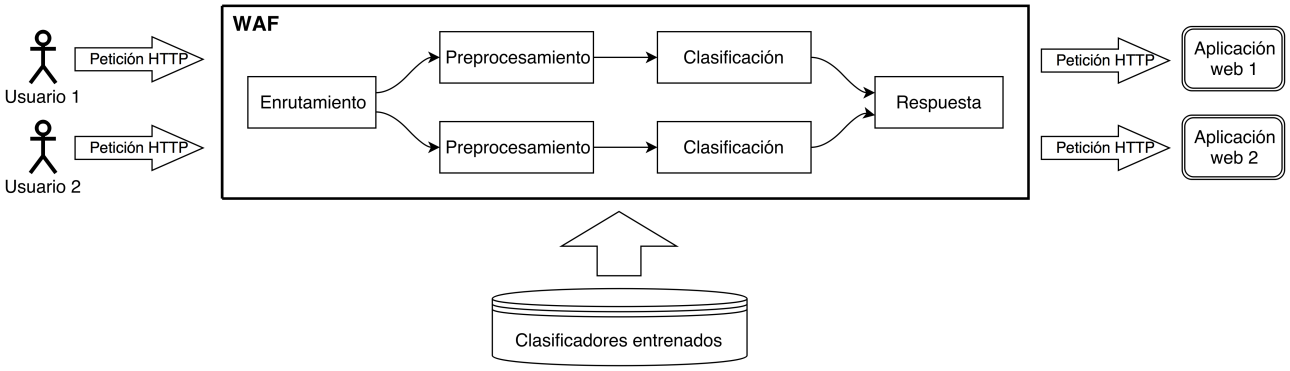

Figura 4.3: Arquitectura de OCS-WAF en la fase de detección.

4.4.1. Paso de enrutamiento

Como se puede observar en la Figura 4.3, 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.

4.4.2. 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 calculados para el grupo G_i correspondiente.

4.4.3. 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 $g_i(\vec{x})$ presentada en la Ecuación 3.15 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).

4.4.4. 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. Por ejemplo, ciertos ambientes podrían preferir bloquear todas las anomalías para evitar posibles ataques. Esto podría causar que peticiones normales incorrectamente clasificadas no lleguen a su destino (se producen falsos positivos). En cambio, otros ambientes podrían optar por solamente lanzar alertas y no bloquear las anomalías para evitar que peticiones normales erróneamente clasificadas como anómalas (justamente los falsos positivos mencionados) no lleguen a su destino y afecten la experiencia de usuarios legítimos.

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, para que los administradores estén enterados al instante y no sea necesario esperar la inspección de los *logs* para percatarse de distintos tipos de incidentes de seguridad.

4.5. Limitaciones de la implementación

Nuestra implementación busca ser funcional y sencilla. Se trata de una prueba de concepto, y por este motivo no nos hemos enfocado en obtener una aplicación terminada que incluya todas las funcionalidades necesarias para ser utilizada directamente en ambientes de producción. A continuación, mencionamos las limitaciones que están presentes en el WAF que implementamos en el marco de este trabajo:

- OCS-WAF no cuenta con un panel de administración para visualizar el estado del mismo o realizar cambios en las configuraciones. Se debe realizar el inicio del detector mediante un *script*, utilizando parámetros para seleccionar las opciones de configuración.
- OCS-WAF podría ser utilizado para la recolección de peticiones normales para la fase de entrenamiento. Los procesos de detección pueden ser deshabilitados indicando la configuración deseada al momento de inicialización del WAF, pero nuestra implementación no cuenta todavía con mecanismos para almacenar las peticiones en memoria secundaria.
- Para el entrenamiento de los clasificadores, OCS-WAF no cuenta todavía con un método automático para la selección de los valores para los parámetros ν_i y γ_i, de forma que estos deben ser especificados por los administradores del detector.

En el siguiente capítulo presentamos las pruebas que realizamos con nuestra implementación y los resultados que obtuvimos mediante las mismas.

Capítulo 5

Pruebas y resultados

En este capítulo presentamos las pruebas que realizamos con OCS-WAF. Primeramente describimos los conjuntos de datos de prueba utilizados, luego explicamos las pruebas realizadas y finalmente comparamos los resultados obtenidos con otros trabajos relacionados.

5.1. 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. Pero esto no es una tarea fácil, ya que muchos conjuntos de datos tienen ciertas falencias que los hacen poco útiles para nuestras pruebas (Torrano-Giménez, 2015).

5.1.1. Características deseadas para los conjuntos de datos

Según (Torrano-Giménez, 2015) los conjuntos de datos utilizados en las investigaciones sobre WAF deberían cumplir ciertos criterios para ser de utilidad para la comunidad científica. Algunas de esas características son:

- Deberían ser públicos para que investigadores puedan comparar resultados.
- Deberían contener tráfico HTTP, ya que se trata de un WAF.
- Deberían contener datos etiquetados según sean normales o anómalas.
- Deberían incluir ataques novedosos y relevantes.

Los trabajos relacionados sobre WAF utilizan distintos conjuntos de datos para sus pruebas, pero la mayoría de esos conjuntos no cumple una o varias de las características mencionadas anteriormente (Torrano-Giménez, 2015). Encontramos dos conjuntos de datos adecuados según los criterios mencionados; fueron creados por el Consejo Superior de Investigaciones Científicas (CSIC) de España y publicados como CSIC 2010 (Torrano-Giménez et al., 2010) y CSIC TORPEDA 2012 (Torrano-Giménez et al., 2012). Varias investigaciones relacionadas ya utilizaron estos conjuntos, como por ejemplo (Parhizkar and Abadi, 2015) y (Torrano-Giménez, 2015), y así podemos comparar resultados con dichos trabajos.

5.1.2. Descripción de los conjuntos de datos utilizados

Los dos conjuntos de datos del CSIC contienen peticiones HTTP hechas a una aplicación sencilla de comercio electrónico, simulando distintos escenarios como, por ejemplo, registro de clientes y compra de productos. Una descripción más detallada sobre el proceso de creación de estos conjuntos se puede encontrar en el trabajo (Torrano-Giménez, 2015).

El conjunto CSIC 2010 (Torrano-Giménez et al., 2010) consiste de tres archivos: dos archivos de 20 MB con tráfico HTTP normal, que contienen alrededor de 72 000 peticiones, y un archivo de 15 MB con tráfico anómalo, que contiene cerca de 25 000 peticiones. Este último archivo contiene varios tipos de ataques, además de anomalías que no necesariamente son ataques pero representan peticiones con datos no esperados.

Considerando que nuestro WAF entrena un clasificador por cada combinación de URL y método HTTP, nosotros agrupamos todas las peticiones de los tres archivos por este criterio. De esta manera obtuvimos más de 1 600 grupos distintos, pero notamos que muchos de ellos tenían solamente peticiones normales o solamente peticiones anómalas. Esos casos no nos sirven para las pruebas, ya que necesitamos datos normales para el entrenamiento y posteriormente anomalías para validar que el WAF detecta correctamente estas últimas. Por este motivo decidimos utilizar solamente aquellos grupos que tienen más de 100 peticiones normales y también más de 100 peticiones anómalas. Así obtuvimos 16 grupos, alcanzando un total de 32 000 peticiones normales y 19 160 anómalas. En la Tabla 5.1 se pueden ver en detalle los grupos de peticiones que utilizamos para nuestras pruebas, incluyendo un identificador corto que le asignamos a cada uno de los grupos para la mejor visualización de los resultados.

El conjunto de datos CSIC TORPEDA 2012 (Torrano-Giménez et al., 2012) consiste de ocho archivos: un archivo de 8 MB con tráfico HTTP normal, que contiene alrededor de 8 000 peticiones, y siete archivos totalizando más de 60 MB de tráfico anómalo, que contienen más de 65 000 peticiones con ataques y también anomalías no maliciosas.

ID	Conjunto de datos CSIC	Método HTTP y URL		Cant. parám.	Peticiones normales	Peticiones anómalas
c00	2010	GET	/tienda1/miembros/editar.jsp	13	2 000	1 362
c01	2010	POST	/tienda1/miembros/editar.jsp	13	2 000	1 362
c02	2010	GET	/tienda1/publico/anadir.jsp	5	2 000	1 380
c03	2010	POST	/tienda1/publico/anadir.jsp	5	2 000	1 380
c04	2010	GET	/tienda1/publico/autenticar.jsp	5	2 000	1 361
c05	2010	POST	/tienda1/publico/autenticar.jsp	5	2 000	1 361
c06	2010	GET	/tienda1/publico/caracteristicas.jsp	1	2 000	954
c07	2010	POST	/tienda1/publico/caracteristicas.jsp	1	2 000	954
c08	2010	GET	/tienda1/publico/entrar.jsp	1	2 000	897
c09	2010	POST	/tienda1/publico/entrar.jsp	1	2 000	897
c10	2010	GET	/tienda1/publico/pagar.jsp	3	2 000	1 343
c11	2010	POST	/tienda1/publico/pagar.jsp	3	2 000	1 343
c12	2010	GET	/tienda1/publico/registro.jsp	13	2 000	1 364
c13	2010	POST	/tienda1/publico/registro.jsp	13	2 000	1 364
c14	2010	GET	/tienda1/publico/vaciar.jsp	1	2 000	919
c15	2010	POST	/tienda1/publico/vaciar.jsp	1	2 000	919
t00	2012	POST	/tienda1/miembros/editar.jsp	12	5 608	10 121
t01	2012	POST	/tienda1/publico/registro.jsp	12	2 522	13 163
	Total				40 130	42 444

Tabla 5.1: Peticiones seleccionadas de los conjuntos de datos para nuestras pruebas, agrupadas por método HTTP y URL.

Realizando de vuelta la agrupación de las peticiones obtuvimos 72 grupos distintos, pero también había casos con solamente peticiones normales o solamente anomalías. Después de seleccionar únicamente aquellos con más de 100 peticiones normales y también más de 100 peticiones anómalas, nos quedamos con 2 grupos, alcanzando un total de 8 130 peticiones normales y 23 284 anómalas. En la ya mencionada Tabla 5.1 se pueden ver los detalles de estos grupos también.

5.2. Análisis de la eficacia de detección

Realizamos pruebas para analizar la eficacia de detección de OCS-WAF, utilizando los conjuntos de datos presentados. Con el fin de agilizar el proceso de extracción de *features* y realizar la clasificación de forma más eficiente, para estas pruebas construimos una versión reducida del WAF que no incluye recepción y envío de peticiones, sino que obtiene las peticiones de un archivo. El código se encuentra en el directorio `/tests/detection_efficacy/`.

Cabe volver a mencionar que nuestra implementación no contiene todavía un mecanismo automático para la selección de los valores para los parámetros ν_i y γ_i de los clasificadores. Para estas pruebas, realizamos una búsqueda en el rango $[0{,}0001; 0{,}1]$ para ambos parámetros y seleccionamos los valores que obtuvieron los mejores resultados en la fase de detección para cada grupo de peticiones. Cada búsqueda fue realizada tres veces, utilizando cada vez un subconjunto distinto de peticiones para el entrenamiento, con el fin de evitar resultados dependientes de la selección particular de muestras. Con esto metodología utilizada mostramos que existen valores para los parámetros ν_i y γ_i de los clasificadores con los cuales se puede obtener los resultados de detección que son presentados a continuación.

		Clasificación real	
		N Negatives (peticiones normales)	P Positives (peticiones anómalas)
Detección obtenida	Negatives (peticiones detectadas como normales)	TN True Negatives (peticiones normales detectadas correctamente como normales)	FN False Negatives (peticiones anómalas detectadas incorrectamente como normales)
	Positives (peticiones detectadas como anómalas)	FP False Positives (peticiones normales detectadas incorrectamente como anómalas)	TP True Positives (peticiones anómalas detectadas correctamente como anómalas)

Figura 5.1: Matriz de confusión de los posibles resultados de clasificación.

Para la cuantificación de las pruebas, tomamos en cuenta los distintos resultados de clasificación que se puede obtener en la terminología de IDS, según la matriz de confusión ilustrada en la Figura 5.1. Se puede observar que la cantidad total de peticiones en la fase de detección se compone de las peticiones normales (N, o muestras negativas) y las 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 peticiones no pertenecientes a la clase conocida de los normales, 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 pertenecientes a los mensajes normales, resultando en falsos negativos (FN). Estas cantidades y las relaciones entre ellas también pueden ser expresadas como se puede observar en la Ecuación 5.1.

$$\text{Total de muestras} = \text{N} + \text{P} \ , \quad \text{N} = \text{TN} + \text{FP} \ , \quad \text{P} = \text{TP} + \text{FN} \tag{5.1}$$

Para el análisis de nuestros resultados utilizamos varias métricas que son fórmulas agregadas de estas cantidades vistas.

- *True Positives Rate* (TPR): Esta métrica indica la fracción de peticiones anómalas (muestras positivas) que fue detectada correctamente; tiene un rango $[0; 1]$. El 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.

- *False Positives Rate* (FPR): Esta métrica indica la fracción de peticiones normales (muestras negativas) que fue detectada equivocadamente como anomalías; tiene un rango $[0; 1]$. El 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.

- F_1-*score*: Esta métrica busca incorporar las dos métricas anteriores en un solo indicador; tiene también un rango $[0; 1]$. El valor debería ser también cercano a 1 en los resultados producidos por el WAF 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 con las expresiones que podemos observar en la Ecuación 5.2.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}}\ , \quad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}}\ , \quad \mathrm{F}_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{5.2}$$

En el análisis de la eficacia de detección de OCS-WAF nos enfocamos en cuatro aspectos; las mejoras obtenidas por incluir el escalamiento de los *features*, las mejoras que produce el análisis de los valores de los parámetros en las peticiones, la influencia de la cantidad de peticiones utilizadas para el entrenamiento y finalmente la influencia de peticiones anómalas presentes en los datos de entrenamiento.

5.2.1. Mejoras obtenidas por el escalamiento de *features*

Como ya explicamos en el Capítulo 2, después de la extracción de *features* realizamos un escalamiento para obtener promedio cercano a 0 y varianza cercana a 1 para cada *feature* de los datos de entrenamiento. Según la teoría, este preprocesamiento ayuda a mejorar la capacidad de clasificación del One-Class SVM (Rieck, 2009), y con los resultados de esta prueba logramos comprobar esto.

ID	Sin escalamiento			Con escalamiento		
	TPR	FPR	F_1-*score*	TPR	FPR	F_1-*score*
c00	0,85 ± 0,01	0,31 ± 0,01	0,73 ± 0,00	0,71 ± 0,01	0,05 ± 0,00	0,80 ± 0,00
c01	0,85 ± 0,00	0,32 ± 0,00	0,73 ± 0,00	0,72 ± 0,01	0,05 ± 0,01	0,80 ± 0,00
c02	0,93 ± 0,12	0,13 ± 0,10	0,88 ± 0,01	1,00 ± 0,00	0,03 ± 0,01	0,98 ± 0,01
c03	0,87 ± 0,12	0,08 ± 0,12	0,88 ± 0,01	1,00 ± 0,00	0,03 ± 0,00	0,98 ± 0,00
c04	0,97 ± 0,01	0,25 ± 0,01	0,83 ± 0,01	0,91 ± 0,01	0,01 ± 0,00	0,95 ± 0,01
c05	0,96 ± 0,01	0,26 ± 0,01	0,82 ± 0,01	0,92 ± 0,01	0,01 ± 0,00	0,95 ± 0,00
c06	0,99 ± 0,00	0,08 ± 0,06	0,92 ± 0,05	0,99 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c07	0,99 ± 0,01	0,11 ± 0,01	0,88 ± 0,01	0,99 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c08	0,99 ± 0,02	0,01 ± 0,00	0,98 ± 0,01	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c09	0,98 ± 0,03	0,01 ± 0,01	0,99 ± 0,02	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c10	1,00 ± 0,00	0,14 ± 0,03	0,90 ± 0,02	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c11	1,00 ± 0,00	0,16 ± 0,02	0,90 ± 0,01	1,00 ± 0,00	0,01 ± 0,00	0,99 ± 0,00
c12	0,86 ± 0,01	0,31 ± 0,02	0,75 ± 0,01	0,74 ± 0,00	0,05 ± 0,01	0,81 ± 0,01
c13	0,86 ± 0,01	0,32 ± 0,01	0,74 ± 0,00	0,74 ± 0,00	0,05 ± 0,01	0,81 ± 0,01
c14	0,96 ± 0,04	0,01 ± 0,01	0,97 ± 0,02	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c15	0,94 ± 0,01	0,01 ± 0,01	0,96 ± 0,00	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
t00	1,00 ± 0,00	0,07 ± 0,04	0,98 ± 0,01	0,99 ± 0,01	0,06 ± 0,04	0,98 ± 0,00
t01	0,99 ± 0,01	0,09 ± 0,10	0,99 ± 0,01	1,00 ± 0,00	0,09 ± 0,06	0,99 ± 0,01
	0,94 ± 0,07	0,15 ± 0,12	0,88 ± 0,09	0,93 ± 0,11	0,03 ± 0,03	0,95 ± 0,08

Tabla 5.2: Resultados de detección que muestran las mejoras obtenidas por el escalamiento de *features*, incluyendo el análisis de valores de parámetros.

En la Tabla 5.2 se puede observar los resultados de la clasificación sin y con escalamiento. Para ambos casos usamos 1500 peticiones para el entrenamiento de cada grupo (cerca del 75 % de las muestras) e incluimos los *features* del análisis de valores de parámetros. Al observar los resultados promedios de los grupos de peticiones (la última fila de la tabla), se puede ver que con el escalamiento el F_1-*score* aumenta de 0,88 y 0,95, ya que a pesar de que el TPR promedio baja de 0,94 a 0,93, el FPR mejora considerablemente de 0,15 a 0,03.

Estos resultados pueden ser observados también en forma gráfica en la Figura 5.2, donde se compara el F_1-*score* obtenido en el proceso de detección sin y con escalamiento para cada uno de los 18 grupos de peticiones.

De esta forma, los resultados nos confirman que el escalamiento de los *features* mejora la capacidad de clasificación de nuestro One-Class SVM para la detección de anomalías en peticiones HTTP, al menos para el conjunto de datos que utilizamos.

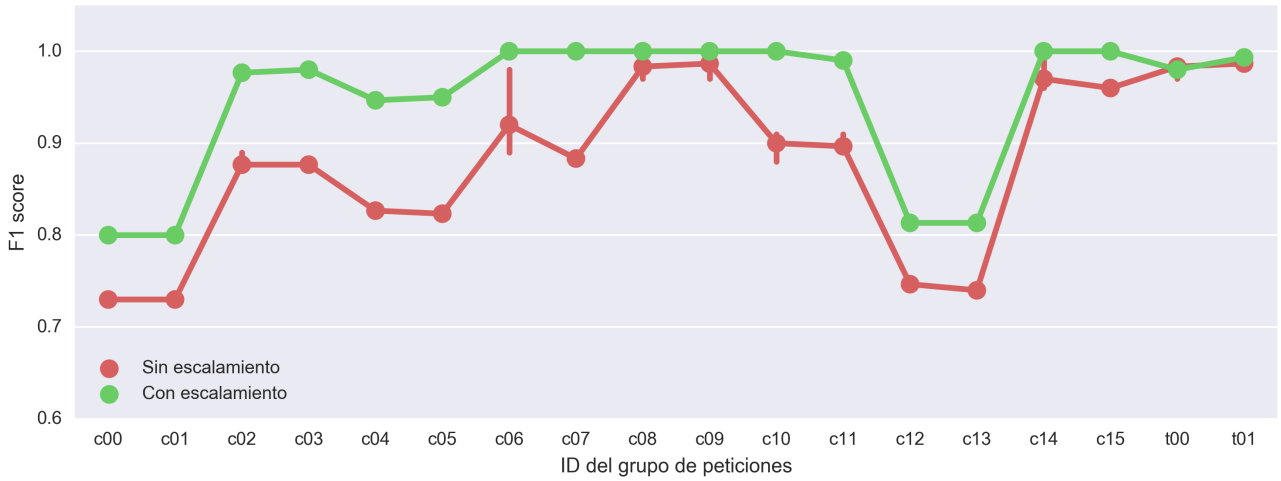

Figura 5.2: Resultados de detección que muestran las mejoras obtenidas por el escalamiento de *features*, incluyendo el análisis de valores de parámetros.

5.2.2. Mejoras obtenidas por el análisis de valores de parámetros

Nuestros procesos de extracción de *features* en primer lugar analizan la petición completa, que resulta en una cantidad fija de *features* que se obtiene para cada petición. Adicionalmente, estos procesos analizan también los valores de los parámetros del *query string* y del cuerpo de las peticiones, lo que resulta en *features* adicionales cuya cantidad depende de la cantidad de parámetros de las peticiones de cada grupo. En esta prueba mostramos que estos *features* adicionales pueden mejorar la detección de OCS-WAF.

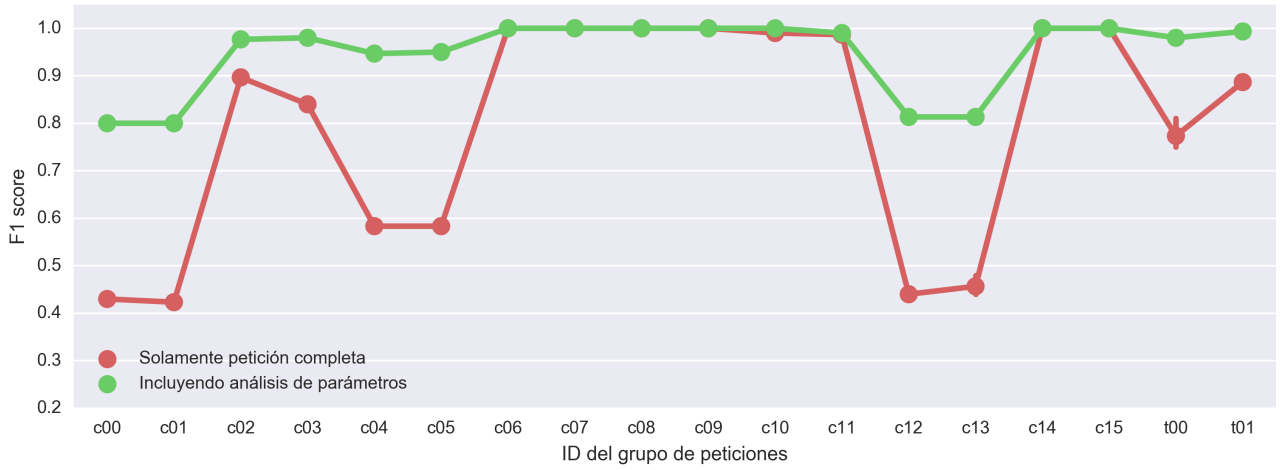

Figura 5.3: Resultados de detección que muestran las mejoras obtenidas con el análisis de valores de parámetros, incluyendo el escalamiento de *features*.

En la Tabla 5.3 y también en la Figura 5.3 se puede observar los resultados de la clasificación usando solamente la petición completa, comparado con los resultados obtenidos incluyendo el análisis de valores de parámetros. Para ambos casos usamos 1500 peticiones para el entrenamiento de cada grupo (cerca del 75 % de las muestras) e incluimos el escalamiento de *features*. En los resultados promedios de todos los grupos (la última fila de la tabla) se puede ver que con el análisis de parámetros el TPR mejora de 0,77 a 0,93, el FPR mejora de 0,11 a 0,03 y el F_1-*score* mejora de 0,79 a 0,95.

ID	Solamente petición completa			Incluyendo análisis de parámetros		
	TPR	FPR	F_1-*score*	TPR	FPR	F_1-*score*
c00	0,32 ± 0,00	0,10 ± 0,01	0,43 ± 0,00	0,71 ± 0,01	0,05 ± 0,00	0,80 ± 0,00
c01	0,31 ± 0,01	0,10 ± 0,01	0,42 ± 0,01	0,72 ± 0,01	0,05 ± 0,01	0,80 ± 0,00
c02	0,89 ± 0,03	0,07 ± 0,05	0,90 ± 0,01	1,00 ± 0,00	0,03 ± 0,01	0,98 ± 0,01
c03	0,82 ± 0,01	0,10 ± 0,01	0,84 ± 0,00	1,00 ± 0,00	0,03 ± 0,00	0,98 ± 0,00
c04	0,83 ± 0,30	0,70 ± 0,52	0,58 ± 0,01	0,91 ± 0,01	0,01 ± 0,00	0,95 ± 0,01
c05	0,65 ± 0,30	0,40 ± 0,52	0,58 ± 0,01	0,92 ± 0,01	0,01 ± 0,00	0,95 ± 0,00
c06	0,99 ± 0,00	0,00 ± 0,00	1,00 ± 0,00	0,99 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c07	0,99 ± 0,00	0,00 ± 0,00	1,00 ± 0,00	0,99 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c08	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c09	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c10	0,99 ± 0,00	0,01 ± 0,01	0,99 ± 0,00	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c11	0,99 ± 0,01	0,01 ± 0,01	0,99 ± 0,01	1,00 ± 0,00	0,01 ± 0,00	0,99 ± 0,00
c12	0,33 ± 0,00	0,10 ± 0,01	0,44 ± 0,00	0,74 ± 0,00	0,05 ± 0,01	0,81 ± 0,01
c13	0,35 ± 0,04	0,13 ± 0,05	0,46 ± 0,02	0,74 ± 0,00	0,05 ± 0,01	0,81 ± 0,01
c14	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
c15	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00	1,00 ± 0,00	0,00 ± 0,00	1,00 ± 0,00
t00	0,67 ± 0,05	0,10 ± 0,08	0,77 ± 0,03	0,99 ± 0,01	0,06 ± 0,04	0,98 ± 0,00
t01	0,82 ± 0,01	0,10 ± 0,01	0,89 ± 0,01	1,00 ± 0,00	0,09 ± 0,06	0,99 ± 0,01
	0,77 ± 0,28	0,11 ± 0,22	0,79 ± 0,23	0,93 ± 0,11	0,03 ± 0,03	0,95 ± 0,08

Tabla 5.3: Resultados de detección que muestran las mejoras obtenidas con el análisis de valores de parámetros, incluyendo el escalamiento de *features*.

De esta forma, los resultados obtenidos muestran la utilidad del análisis de valores de los parámetros de las peticiones. A parte eso, se puede notar una reducción de la varianza en las tres métricas, lo que indica que se redujeron también las diferencias entre los resultados al utilizar distintos subconjuntos de peticiones para el entrenamiento. Esto es un indicador de robustez de OCS-WAF, mostrando que con el análisis de parámetros es menos susceptible a cambios por la elección de peticiones de entrenamiento.

5.2.3. Influencia de la cantidad de peticiones de entrenamiento

Para que OCS-WAF pueda realizar un buen trabajo de detección, es necesario que los datos de entrenamiento cubran la mayor cantidad posible de situaciones que se puedan dar posteriormente en la fase de detección. Esto puede significar que se tenga que pasar una gran cantidad de datos al WAF para realizar el entrenamiento. Pero el simple hecho de aumentar la cantidad de peticiones para esta fase no siempre es la mejor solución, ya que eso aumenta el tiempo que dura la fase de entrenamiento y además no nos asegura que se cubran realmente todos los casos normales posibles.

Los conjuntos de datos de prueba que utilizamos nos proveen con 2 000 peticiones normales en la mayoría de los grupos, como ya se mostró en la Tabla 5.1. Realizamos una prueba para analizar los resultados que se obtienen al utilizar distintas cantidades de esos datos normales para el entrenamiento, siempre incluyendo el escalamiento de *features* y el análisis de valores de parámetros.

Cantidad de peticiones de entrenamiento	Promedio de los 18 grupos		
	TPR	FPR	F_1-*score*
200	0,92 ± 0,11	0,07 ± 0,06	0,92 ± 0,09
500	0,94 ± 0,09	0,06 ± 0,06	0,93 ± 0,09
1 000	0,93 ± 0,11	0,03 ± 0,03	0,94 ± 0,08
1 500	0,93 ± 0,11	0,03 ± 0,03	0,95 ± 0,08

Tabla 5.4: Resultados de detección que muestran la influencia de la cantidad de peticiones de entrenamiento, incluyendo el escalamiento de *features* y el análisis de valores de parámetros.

Como se puede observar en la Tabla 5.4, utilizamos 200, 500, 1 000 y 1 500 peticiones para esta prueba, que equivale a 10 %, 25 %, 50 % y 75 % de los datos disponibles en la mayoría de los grupos. Los mejores resultados obtuvimos 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. Por esta razón también utilizamos 1500 peticiones para el entrenamiento en las pruebas presentadas en las secciones anteriores.

5.2.4. Influencia de anomalías en los datos de entrenamiento

Un desafío en ambientes reales es asegurar que los datos de entrenamiento solamente contengan peticiones HTTP normales y que no haya ninguna anomalía entre ellas. La presencia de ruido (es decir, anomalías) en los datos de entrenamiento puede reducir la capacidad de

clasificación del One-Class SVM, ya que esas anomalías influyen en la posición de la superficie separadora y consecuentemente esto llevará a resultados diferentes en la posterior fase de detección. Como ya presentamos en el capítulo Capítulo 3, el clasificador en cuestión tiene la capacidad de dejar ciertas muestras al mismo lado del hiperplano que el origen durante la fase de entrenamiento, de manera que cuenta con cierta capacidad para reducir la influencia de peticiones anómalas (que son muestras positivas o ataques) que se encuentren dentro de los datos de entrenamiento. En esta prueba buscamos descubrir cómo esto se refleja en el desempeño de detección de OCS-WAF.

Realizamos una prueba para analizar en qué manera el One-Class SVM se ve afectado en su capacidad de clasificación con la presencia de peticiones anómalas en los datos de entrenamiento. Para eso, incluimos una cierta cantidad de anomalías dentro de las peticiones de entrenamiento para llegar a 1 %, 5 % y 10 % de anomalías. Acá volvimos a utilizar 200, 500, 1 000 y 1 500 peticiones para el entrenamiento, siempre incluyendo el escalamiento de *features* y el análisis de valores de parámetros.

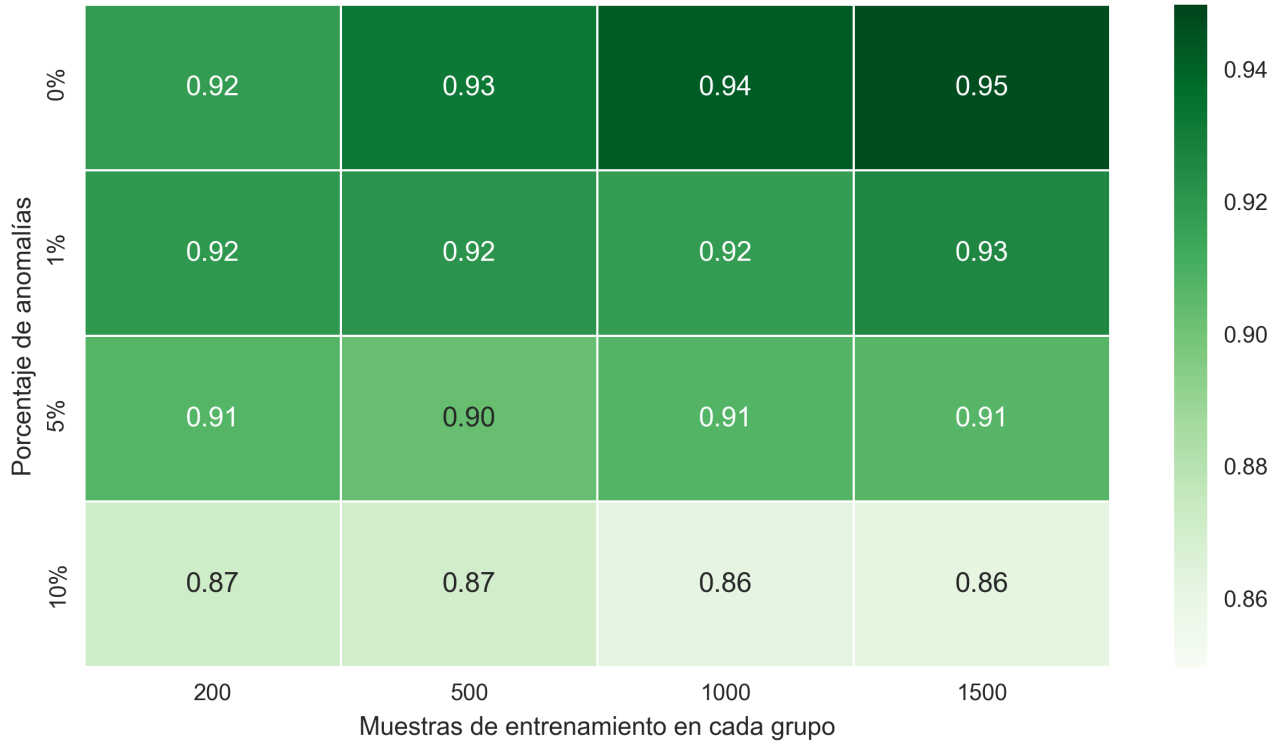

Figura 5.4: Mapa de calor de los resultados de detección, en términos del F_1-*score*, que muestran la influencia de la cantidad de peticiones y el porcentaje de anomalías en el entrenamiento, incluyendo el escalamiento de *features* y el análisis de valores de parámetros.

La Figura 5.4 muestra un mapa de calor (*heatmap*) de los resultados de detección de esta prueba. Las columnas contienen las distintas cantidad de peticiones utilizadas para el entrenamiento, las filas representan diferentes porcentajes de anomalías que se incluyeron en los datos de entrenamiento y los valores en cada celda indican el F_1-*score* promedio de los 18 grupos que se obtuvo con esa configuración. Las celdas con colores más oscuros contienen números mayores, lo que indica resultados mejores.

Como se puede ver en nuestros resultados, el F_1-*score* sufre una degradación gradual conforme va aumentando el porcentaje de anomalías, aunque es una reducción poco significativa para 1 % de anomalías. Esta degradación de la detección se da de forma similar para todas las cantidad de peticiones utilizadas para el entrenamiento, aunque el caso más pronunciado se da para la mayor cantidad de peticiones. Esto se debe a que ya se tiene mayor cantidad de anomalías presentes, por ejemplo con 10 % de 1 500 peticiones ya hay 150 muestras anómalas que ejercen influencia sobre la posición y forma de la superficie separadora de los clasificadores, lo que se ve reflejado en los resultados.

Cabe resaltar que un 10 % de anomalías de entrenamiento ya es un porcentaje muy elevado de ruido. En ese caso se debería utilizar alguna estrategia adicional para obtener un conjunto de datos de entrenamiento más limpio, como por ejemplo una clasificación previa con aprendizaje no supervisado, con el fin de reducir el impacto negativo sobre la eficacia de detección del WAF.

De esta forma, los resultados obtenidos en esta prueba 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.

5.3. 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. Si este tiempo de respuesta aumenta de gran manera debido al uso del WAF, las aplicaciones podrían quedar inutilizables para el usuario final, significando que nuestra implementación no puede ser usada para detección en tiempo real. Estamos conscientes de que las mediciones realizadas durante esta prueba no necesariamente reflejan un ambiente real, pero sí nos permiten establecer el orden de magnitud de la variación que introduce OCS-WAF en el tiempo de respuesta de las aplicaciones protegidas.

Para esta prueba, nosotros 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. Posteriormente 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.

El código de todas las partes utilizadas en esta prueba esta en nuestro repositorio público, en el directorio `/tests/waf_speed/`. Ejecutamos todos los procesos en una misma máquina, buscando reducir de esta manera diferentes variables presentes en la red y concentrarnos en el retraso introducido por el uso del WAF. La máquina utilizada tiene una plataforma *Ubuntu 14.04* de 64 *bits*, con un procesador *Intel i5* de 1,7 GHz y 8 GB de memoria RAM.

Configuración	Tiempo de respuesta promedio (milisegundos)
Sin WAF	4,8 ± 2,2
Con WAF pero sin detección	6,4 ± 1,7
Con WAF y con detección	10,1 ± 2,6

Tabla 5.5: Resultados de la prueba de tiempo de respuesta de las aplicaciones protegidas.

En la Tabla 5.5 se pueden ver los resultados de esta prueba. Para cada una de las tres configuraciones cronometramos 160 peticiones pertenecientes a distintos grupos de método y URL, de las cuales la mitad fueron peticiones normales y la otra mitad anomalías. No pudimos percibir diferencias significativas de tiempo entre peticiones normales y anómalas, ni tampoco entre peticiones de distintos grupos (con distintas cantidades de parámetros). Posibles variaciones están contenidas dentro de la desviación estándar indicada en la tabla mencionada. Los resultados se mantuvieron constantes en repeticiones de la misma prueba.

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 (InternetWeatherMap, 2017) 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 web protegidas. Para aplicaciones dentro de una red interna de alta velocidad, el retraso podría llegar a ser perceptible, pero nuestra implementación fue hecha en primer lugar para aplicaciones accesibles a través de Internet, como fue explicado en la introducción en el Capítulo 1.

5.4. 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. De la misma manera que en la prueba anterior, estamos conscientes de que las mediciones realizadas durante esta prueba no necesariamente reflejan un ambiente real con un WAF optimizado (incluso utilizando otro lenguaje de programación), pero sí nos permiten establecer el orden de magnitud del tiempo de entrenamiento de nuestra implementación y la relación entre este tiempo y la cantidad de peticiones que fueron utilizadas.

El código para esta prueba también esta en nuestro repositorio público, en el directorio `/tests/training_duration/`. Volvimos a utilizar la misma máquina para ejecutar las pruebas, que tiene una plataforma *Ubuntu 14.04* de 64 *bits*, con un procesador *Intel i5* de 1,7 GHz y 8 GB de memoria RAM.

Cantidad de peticiones	Tiempo promedio por petición (segundos)	Tiempo máximo por petición (segundos)	Tiempo total máximo (segundos)
10	0,0013 ± 0,0004	0,0020	0,0195
100	0,0011 ± 0,0004	0,0017	0,1724
1 000	0,0010 ± 0,0004	0,0017	1,7126
10 000	0,0011 ± 0,0004	0,0018	17,9132
100 000	0,0020 ± 0,0005	0,0070	703,9531

Tabla 5.6: Resultados de la prueba de tiempo de entrenamiento.

La Tabla 5.6 muestra los resultados de esta prueba, indicando el tiempo por petición que se alcanzó para las diferentes cantidades utilizadas para el entrenamiento. Para cada una de estas cantidades, realizamos entrenamientos con cada uno los grupos de URL y método HTTP. Cabe mencionar que el tiempo de entrenamiento de cada clasificador está influenciado por la cantidad de *features* extraídos en cada uno de estos grupos de peticiones y debido a esto el tiempo medido puede ser diferente para cada grupo. La tabla de resultados muestra el tiempo promedio y también el tiempo máximo; este último puede ser utilizada a modo de establecer un límite superior de la duración del entrenamiento.

En la tabla mencionada podemos observar 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 tardar unos 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.

5.5. Comparación con trabajos relacionados

En esta sección queremos comparar 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. En la Tabla 5.7 se muestra un resumen de la comparación de resultados de nuestra implementación con trabajos de otros investigadores.

Sistema de detección	Observaciones	TPR	FPR	F_1-*score*
OCS-WAF (el presente trabajo)	One-Class SVMs con cantidad variable de *features*	0,93	0,03	0,95
ModSecurity (Giménez, 2015)	popular detector por firmas de ataques	0,56	0,00	0,71
HTTP-WS-AD (Giménez, 2015)	modelos estadísticos con agregación y cascada	0,99	0,02	0,99
(Torrano-Giménez, 2015)	promedio de cuatro tipos distintos de árboles de decisión	0,95	0,05	-
OC-WAD (Parhizkar and Abadi, 2015)	conjunto de One-Class SVMs con cantidad fija de *features*	0,96	0,03	-

Tabla 5.7: Comparación de resultados de OCS-WAF con otros trabajos que utilizan también el conjunto de datos CSIC 2010.

Antes de analizar otras propuestas de detección de anomalías, investigamos sobre IDS basados en firmas de ataques (no realizan detección de anomalías) para tener una referencia sobre este tipo de sistemas de detección. Una implementación popular de un WAF basados en firmas es *ModSecurity*. En (Giménez, 2015) el autor aplicó este detector al conjunto de datos CSIC 2010, utilizando las firmas de ataques que el mismo trae por defecto. Así obtuvo un TPR de 0,56, un FPR de 0,00 y un F_1-*score* de 0,71. Una fortaleza de este detector es que no incurre en falsos positivos (peticiones normales detectadas como ataques). Pero nuestra implementación obtiene una detección significativamente mejor que este popular detector basado en firmas de ataques, como se puede ver en los resultados de la ya mencionada Tabla 5.7.

En (Giménez, 2015) se presenta un detector de anomalías llamado *HTTP-WS-AD*. En este trabajo se utilizan varios modelos estadísticos y se realizan pruebas con varios esquemas de agregación y cascada de modelos para mejorar los resultados de la detección de peticiones HTTP anómalas. El autor reporta haber obtenido un TPR de 0,99, un FPR de 0,02 y un F_1-*score* de 0,99 para el conjunto de datos CSIC 2010. Cada una de estos tres resultados es levemente superior a los números logrados por OCS-WAF, con la notable diferencia de que este trabajo utiliza métodos estadísticos mientras que nuestra implementación emplea herramientas del área de ML. Además, Giménez también realizó mediciones del impacto de su detector en el tiempo de respuesta de las aplicaciones web protegidas, reportando un retraso adicional promedio de 50 milisegundos introducido por las tareas de detección. OCS-WAF obtuvo un retraso promedio menor a 10 milisegundos en nuestras pruebas, mostrando una leve ventaja en este aspecto frente a *HTTP-WS-AD*.

En la segunda parte del trabajo (Torrano-Giménez, 2015) se presenta un detector de anomalías que utiliza cuatro tipos de árboles de decisiones para su proceso de detección. Estos árboles son también herramientas de clasificación del área de ML, pero en este caso se trata de clasificadores binarios o de dos clases. Esto significa que se entrena los clasificadores con peticiones normales y anómalas. El trabajo en cuestión reporta haber obtenido un TPR de 0,95 y un FPR de 0,05 sobre el conjunto de datos CSIC 2010. OCS-WAF obtuvo un TPR inferior con 0,93, pero obtuvo un FPR superior de 0,03. Cabe resaltar que el trabajo en cuestión entrena sus clasificadores con ambas clases de peticiones (normales y anomalías), mientras que nosotros entrenamos con peticiones normales únicamente. Creemos que debido a eso, OCS-WAF es más robusto frente a la aparición de nuevos tipos de ataques, ya que no se usan anomalías en el entrenamiento.

En (Parhizkar and Abadi, 2015) se presenta un detector de anomalías llamado *OC-WAD* para peticiones HTTP. Los autores extraen una cantidad fija de *features* de las peticiones y

posteriormente utilizan un conjunto de clasificadores One-Class SVM con *kernel* RBF para realizar la detección. Los distintos clasificadores son entrenados con diferentes subconjuntos de *features*. Además, ellos presentan un algoritmo de enjambres llamado *BeeSnips*, que utilizan para la creación y optimización del conjunto de clasificadores. Trabajando con el conjunto de datos CSIC 2010, los autores reportan haber obtenido un TPR de 0,96 y un FPR de 0,03. Estos resultados son similares a los nuestros; obtuvimos un TPR inferior de 0,93 y un FPR igual de 0,03. Pero en este trabajo nos se brinda mucha información sobre la manera que realizaron sus pruebas; por ejemplo no explican si entrenan conjuntos de clasificadores separados por URL, lo que nos deja dudas sobre cuan comparables son nuestros resultados.

Capítulo 6

Conclusiones

En este capítulo presentamos las conclusiones de nuestro trabajo. Primeramente damos un resumen general del contenido que fue presentado a lo largo de este trabajo, incluyendo el análisis de los resultados que obtuvimos en las pruebas realizadas con el detector OCS-WAF. Después mostramos de qué forma hemos logrado alcanzar los objetivos que nos propusimos para esta investigación. Finalmente, concluimos este trabajo describiendo varios aspectos de nuestra implementación en los cuales pueden realizarse mejoras; estos aspectos podrían dar lugar a trabajos futuros.

6.1. Resumen de la investigación

En esta sección presentamos los aspectos más importantes que fueron descritos en cada uno de los capítulos anteriores, buscando brindar de esta forma un resumen de las ideas principales de este trabajo.

En el Capítulo 1 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. Al final de ese capítulo presentamos los objetivos que nos hemos propuesto para este trabajo de investigación; estos objetivos se centran 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 el Capítulo 2 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 el Capítulo 3 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. 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 conjuntos de datos diferentes.

El Capítulo 4 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`.

El Capítulo 5 presenta las pruebas realizadas con nuestra implementación y expone los resultados obtenidos en cada una de ellas. Utilizamos los conjuntos de datos CSIC 2010 (Torrano-Giménez et al., 2010) y CSIC TORPEDA 2012 (Torrano-Giménez et al., 2012), 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 del Capítulo 5 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.

6.2. Alcance de los objetivos

En esta parte volvemos a presentar los objetivos que nos habíamos propuesto en el Capítulo 1 y mostramos de qué manera logramos alcanzar los mismos a través del trabajo realizado a lo largo de esta investigación. Empezamos describiendo nuestros logros para los objetivos específicos, para después cerrar con el objetivo general.

6.2.1. 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 el Capítulo 2, mientras que en el Capítulo 4 se mostraron los detalles de la implementación de los mismos. Las pruebas experimentales presentadas en el Capítulo 5 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 el Capítulo 2, junto con los clasificadores One-Class SVM que fueron presentados en el Capítulo 3. Los detalles de implementación del OCS-WAF fueron explicados en el Capítulo 4. 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 del Capítulo 5 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 del Capítulo 5 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.

6.2.2. 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.

6.3. Ideas para trabajos futuros

A pesar de concluir exitosamente nuestra investigación, reconocemos que aún hay varios aspectos que pueden ser mejorados o extendidos. A continuación presentamos algunas de estas ideas que podrían dar lugar a trabajos futuros.

- Se podría realizar pruebas con OCS-WAF utilizando otros conjuntos de datos. Esto serviría para verificar y validar que nuestra implementación no está atada solamente a los datos que utilizamos en este trabajo.

- Nuestros resultados aún dejan lugar para mejoras, especialmente el TPR es un poco bajo para algunos casos. Se podría explorar otras características de los mensajes HTTP para extender y mejorar nuestros procesos de extracción de *features*, analizando el impacto que tiene cada característica en la detección.

- 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 otros formatos, por ejemplo datos binario, JSON, XML, entre otros. Adicionalmente, se necesita encontrar o construir un conjunto de datos de prueba que contenga ese tipo de peticiones con datos suficientemente reales.

- 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 (Belshe et al., 2015) y se podría incluir una extensión a nuestra implementación para poder analizar también los mensajes HTTP/2.

- La selección de los valores para los parámetros ν_i y γ_i presenta un gran desafío y OCS-WAF no cuenta todavía con un método automático para esta búsqueda. Se podría explorar métodos para realizar la selección óptima de estos parámetros, para posteriormente incorporar también ese mecanismo descubierto dentro de nuestra implementación.

Referencias

Abadeh, M. S., Mohamadi, H., and Habibi, J. (2011). Design and analysis of genetic fuzzy systems for intrusion detection in computer networks. *Expert Systems with Applications*. 5, 31

Aggarwal, C. C. (2013). *Outlier Analysis*. Springer Publishing Company, Incorporated. 4, 29, 31, 37, 38

Amer, M., Goldstein, M., and Abdennadher, S. (2013). Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*. ACM. 34, 38, 39

Belshe, M., Peon, R., and Thomson, N. (2015). Rfc 7540: Hypertext transfer protocol version 2 – http/2. Technical report, RFC Editor, United States. 76

Berners-Lee, T., Fielding, R., and Masinter, L. (2005). Rfc 3986: Uniform resource identifier (uri): Generic syntax. Technical report, RFC Editor, United States. 11

Buczak, A. L. and Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*. 3, 30, 31

Corchado, E. and Herrero, l. (2011). Neural visualization of network traffic data for intrusion detection. *Applied Soft Computing*. 5, 31

Dobrushin, R. and Prelov, V. (2011). Entropy - encyclopedia of mathematics. `http://www.encyclopediaofmath.org/index.php?title=Entropy&oldid=15099`. Accessed: August-2017. 21

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Rfc 2616: Hypertext transfer protocol – http/1.1. Technical report, RFC Editor, United States. 2, 9, 10

Giménez, J. (2015). Http-ws-ad: Detector de anomalías orientada a aplicaciones web y web services. Universidad Nacional de Asunción. 1, 3, 30, 68, 69

Giménez, J. and Cappo, C. (2015). Http-ws-ad: Anomaly detector oriented to web applications and services. In *Computing Conference (CLEI), 2015 Latin American*. IEEE. 3

InternetWeatherMap (2017). Internet weather map. `https://www.internetweathermap.com/map`. Accessed: August-2017. 67

Khan, S. and Madden, M. (2014). One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*. 4, 5, 7, 30, 31

Khan, S. S. and Madden, M. G. (2009). A survey of recent trends in one class classification. In *Irish Conference on Artificial Intelligence and Cognitive Science*. Springer Berlin Heidelberg. 4, 29, 30, 31, 51

Kruegel, C. and Vigna, G. (2003). Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*. ACM. 2, 3, 5, 6, 11, 13, 18, 23, 30, 46

Kruegel, C., Vigna, G., and Robertson, W. (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks*. 5, 6, 11, 13

Lagadec, P. (2011). Cherryproxy - a filtering http proxy extensible in python. `http://www.decalage.info/python/cherryproxy`. Accessed: July-2017. 47

Nguyen, H. T., Torrano-Gimenez, C., Alvarez, G., Petrović, S., and Franke, K. (2011). Application of the generic feature selection measure in detection of web attacks. In *Computational Intelligence in Security for Information Systems*. Springer, Berlin, Heidelberg. 13, 22, 23

Nguyen, X. N., Nguyen, D. T., and Vu, L. H. (2016). Pocad: A novel pay load-based one-class classifier for anomaly detection. In *Information and Computer Science (NICS), 2016 3rd National Foundation for Science and Technology Development Conference on*. IEEE. 40

Nikulin, M. (2012). Chi-squared test - encyclopedia of mathematics. `http://www.encyclopediaofmath.org/index.php?title=Chi-squared_test&oldid=28552`. Accessed: September-2017. 19

Parhizkar, E. and Abadi, M. (2015). Oc-wad: A one-class classifier ensemble approach for anomaly detection in web traffic. In *2015 23rd Iranian Conference on Electrical Engineering (ICEE)*. IEEE. 3, 30, 31, 32, 33, 34, 35, 40, 56, 68, 69

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*. 48

Perdisci, R., Gu, G., and Lee, W. (2006). 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. 35, 37, 38, 39, 40

Rieck, K. (2009). *Machine learning for application-layer intrusion detection*. PhD thesis, Technischen Universität Berlin. Accessible under: `http://dx.doi.org/10.14279/depositonce-2199`. 9, 25, 35, 36, 41, 59

Robertson, W. (2009). *Detecting and preventing attacks against web applications*. PhD thesis, University of California, Santa Barbara. 1

Scarfone, K. A. and Mell, P. M. (2007). Sp 800-94. guide to intrusion detection and prevention systems (idps). Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States. 2, 44

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*. 5, 31

Sommer, R. and Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 3, 30, 31

Torrano-Giménez, C. (2015). *Study of stochastic and machine learning techniques for anomaly-based web attack detection*. PhD thesis, Universidad Carlos III de Madrid. 2, 3, 4, 5, 9, 29, 30, 31, 44, 55, 56, 68, 69

Torrano-Giménez, C., Pérez, A., and Álvarez, G. (2012). Csic torpeda 2012 http data sets. `http://www.tic.itefi.csic.es/torpeda`. Accessed: July-2017. 56, 72

Torrano-Giménez, C., Pérez Villegas, A., and Álvarez Marañón, G. (2010). Csic 2010 http data sets. `http://www.isi.csic.es/dataset/`. Accessed: July-2017. 56, 72

Tran, Q.-A., Duan, H., and Li, X. (2004). One-class support vector machine for anomaly network traffic detection. *China Education and Research Network (CERNET), Tsinghua University, Main Building*, 310. 41