VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

DETECCIÓN DE ATAQUES DE INYECCIÓN SQL EN APLICACIONES WEB BASADO EN UN CLASIFICADOR DE SIMILITUD DE TEXTOS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA
IVÁN ISRAEL SABIDO CORTÉS

TUTOR
DR. JORGE LUIS ORTEGA ARJONA
FACULTAD DE CIENCIAS, UNAM

MÉXICO, D.F. SEPTIEMBRE, 2015

Dedicatoria

A mi hija Sarah, por quién cada día tiene sentido, porque tuviste que estar tanto tiempo sin la compañía de tu papá. A pesar de ello, cada vez que podíamos reunirnos, aprovechábamos hermosos momentos, en los que tu sola sonrisa me llenaba de ánimos y fuerzas para seguir adelante y no dejarme vencer. Espero comprendas, que todo el sacrificio ha sido para darte un mejor futuro y un día puedas compartir tus logros conmigo.

A mi hijo Elián, tu nacimiento ha sido de mucha alegría y un impulso más para finalizar este trabajo. De igual forma, quiero que sepas que haré todo lo que esté a mi alcance para que si así lo decides, un día puedas dedicar uno o más trabajos como este.

A mi amada esposa, Carolina, que ha sido el impulso durante todo el posgrado y el pilar principal para la culminación de la misma, que con su apoyo constante y amor incondicional ha sido amiga y compañera inseparable, fuente de calma, sabiduría y consejo en todo momento.

A mis padres, Sagrario y Edgar, por darme la vida, una maravillosa formación, por su ternura y todo su amor, y por contagiarme de sus mayores fortalezas. Mamá, tú me das el ejemplo de ser decidido y a levantarme después de cada tropiezo, con la idea de que me espera algo mejor. Papá, me enseñaste a ser paciente y a ver los problemas con la cabeza fría y como situaciones solucionables.

A mis abuelos, Ermilo y Raquel, que a pesar de no estar aquí, sé que están orgullosos de este nuevo logro. Fueron un excelente ejemplo de sacrificio y esfuerzo para mí, ustedes me inspiraron a mejorar cada día y me enseñaron a afrontar mis problemas con optimismo.

A mis suegros, Félix y Lilia, gracias por su cariño, comprensión, apoyo y paciencia. Por quererme y cuidarme como su hijo.

Agradecimientos

Un reconocimiento explícito a la Universidad Nacional Autónoma de México, en particular al Dr. Jorge Luis Ortega Arjona, por su paciencia, apoyo y consejos en todo momento, al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS), a la Dirección de Posgrados y CONACyT por su apoyo y por proporcionarme los fondos necesarios para llevar a cabo esta investigación y la estancia de investigación en la Universidad Complutense de Madrid.

Contenido

Índice de Figuras

Índice de Tablas

Resumen

En este trabajo se hace un análisis en relación a la detección de inyecciones de código SQL, mediante el uso del modelo vectorial y el modelo de Tri-gramas, con el fin de ofrecer o relacionar técnicas de otras áreas en el área de seguridad Web.

Se aborda el impacto que pueden tener estos ataques, así como las diferentes formas en las que se trata de resolver el problema, esto, con el fin de ofrecer recomendaciones y dar una mayor visión al momento de establecer mecanismos de seguridad para evitar este tipo de problemas.

CAPÍTULO I

Introducción

"La máxima seguridad es tu comprensión de la realidad".

H. Stanley Judd

En este capítulo se considera la seguridad en aplicaciones Web como el contexto del problema a resolver. Se presentada una explicación al problema que existe derivado de la inyección de código malicioso, y se propone una solución de cómo manejarlo en un firewall de aplicación Web.

1.1 Contexto

La seguridad en aplicaciones Web se ha hecho indispensable debido al manejo de información sensible y al acceso público que tienen las aplicaciones. El acceso no autorizado a dicha información podría causar pérdidas económicas, incluyendo en ello, hasta la mala reputación de una empresa o gobierno.

Las aplicaciones Web han ido evolucionando debido a que los navegadores ofrecen cada vez más y mejores funcionalidades, tienen un consumo bajo de los recursos, no ocupan mucho espacio y son fáciles de actualizar de manera rápida. Esto hace posible que las aplicaciones Web puedan realizar tareas sencillas y complejas, que no existan problemas de compatibilidad y que su precio resulte ser menor al de las aplicaciones de escritorio. Dos características importantes de las aplicaciones Web y que le han dado fama son: su alta disponibilidad y acceso desde cualquier parte del mundo. Estas características han ayudado a que las empresas,

bancos, gobierno y universidades cambien sus sistemas de escritorio por aplicaciones Web. Sin embargo, de igual forma estas características han atraído a los atacantes para obtener beneficios.

La evolución de las aplicaciones Web, ha producido que su desarrollo sea cada vez más complejo. Hace algunos años era común escuchar la existencia de fallas en los sistemas de protección de los servidores Web, siendo los más frecuentemente utilizados el Apache e IIS de Microsoft, o errores en los lenguajes de programación en las que estaban escritas. Pero es un hecho, que la mayoría de los problemas detectados en aplicaciones Web no son provocados por fallas intrínsecas de ninguna de estas partes. La mayoría de los problemas se deben a la mala programación de los desarrolladores o a la falta de conocimiento en las áreas o elementos de seguridad (UNAM-CERT, 2009).

En el 2013, el proyecto abierto de seguridad en aplicaciones Web (OWASP por sus siglas en inglés) da a conocer la actualización de su lista de los riesgos más serios o críticos que pueden tener las aplicaciones Web. El primer lugar lo ocupan las fallas de inyección, tales como SQL, OS, y LDAP, seguidas de la Pérdida de Autenticación y Gestión de Sesiones (OWASP, 2013). Muchas de estas fallas pueden deberse a la falta de conocimiento de los desarrolladores en materia de seguridad o a la falta de tiempo para el desarrollo del proyecto, o bien a la premura de sacar un nuevo producto al mercado.

En un estudio realizado en el período Agosto, 2013 – Abril, 2014 por la empresa Imperva (Imperva, 2014), se aprecia un incremento del 44% de ataques a las aplicaciones Web en comparación a las realizadas en el período Junio, 2012 – Noviembre, 2012. De igual forma, se observa que las aplicaciones de ventas al por menor son las más atacadas (48.1%), dejando en segundo lugar a las aplicaciones de instituciones financieras (10%). Estos ataques son realizados a las aplicaciones que cuentan con algún tipo de información de los consumidores o

clientes, o bien que contienen información sensible.

Debido a la naturaleza de los ataques y al incremento de éstos, no es suficiente con proteger solo la capa de red del modelo OSI. Esto ha dado motivo para investigar nuevas técnicas y desarrollar nuevas herramientas y métodos para la detección de ataques en la capa de aplicación. Una de estas herramientas desarrollada en software o en Hardware es conocida como firewall de aplicación Web (WAF por sus siglas en inglés).

1.2 Planteamiento del Problema

Como se ha mencionado en la sección anterior, las fallas de inyección han sido valoradas como el riesgo número uno dentro de la publicación de los riesgos más críticos en aplicaciones Web (OWASP, 2013). Dentro de estos tipos de inyección se encuentra una de las más comunes y peligrosas denominada *Inyección SQL*. A pesar de ser una vulnerabilidad conocida y antigua, sigue siendo uno de los principales problemas en las aplicaciones Web desarrolladas por pequeñas y grandes empresas.

En un informe realizado por la empresa Imperva (2014), se observa que las Inyecciones SQL tuvieron un incremento del 10% dentro de los ataques realizados a las aplicaciones Web. De los ataques realizados a las aplicaciones de venta al por menor, el 40% fueron Inyecciones SQL. De igual forma, se sigue encontrando este tipo de vulnerabilidad en sistemas como los Manejadores de Contenidos (CMS por sus siglas en inglés), como ejemplo se tiene el plugin de Wordpress llamado WP-Slimstat, el cual estaba instalado en más de un millón de sistemas (Santillan, 2015), por lo que éstos eran vulnerables a inyecciones SQL.

OWASP (2012) define una inyección SQL de la siguiente forma:

> Un ataque por inyección SQL consiste en la inserción o "inyección" de una consulta

SQL por medio de los datos de entrada desde el cliente hacia la aplicación. Un ataque por inyección SQL exitoso puede leer información sensible contenida en la base de datos del servidor de aplicaciones Web, modificar la información (Insert/ Update/ Delete), ejecutar operaciones de administración sobre dicha base de datos (tal como deshabilitar o detener la base de datos), extraer el contenido de un determinado archivo presente sobre el sistema de archivos del DBMS y en algunos casos emitir comandos al sistema operativo.

Para llevar a cabo este tipo de ataque, por lo general, suelen inyectarse consultas en campos de formularios o en los parámetros de la URL, que no han sido previstos o en el mejor caso validados del lado del servidor por su desarrollador, creando una vulnerabilidad permanente en el sistema. Esta inyección tiene la finalidad de modificar las consultas SQL originales o predefinidas por los desarrolladores.

Este tipo de vulnerabilidades son consideradas críticas y deben evitarse a toda costa. El enfoque que a veces suelen tener los desarrolladores, es simplemente la de reparar la vulnerabilidad, siempre y después de que el sistema haya sido comprometido. Lo anterior puede llevar a perder mucho dinero a la empresa y/o a sus clientes.

Este tipo de vulnerabilidades se presentan debido a que los desarrolladores en ocasiones caen en los siguientes mitos:

- El usuario solamente enviará entradas esperadas – HTML admite el uso de tags que manipulan las entradas a la aplicación, por ejemplo si la aplicación utiliza campos ocultos para enviar información sensible, estos pueden ser fácilmente manipulados desde el cliente. También se debe recordar que si el servicio es público cualquier persona puede acceder a la aplicación y no necesariamente con buenas intenciones.

- La validación puede realizarse únicamente del lado del cliente con JavaScript - si no se efectúa ninguna validación del lado del servidor, cualquier atacante que evite esta validación (existen muchos programas conocidos como proxys, uno muy famoso es un plugin de Firefox llamado Tamper Data) tendrá acceso a modificar cualquier parámetro o información enviada al servidor.
- El uso de firewalls de red es suficiente - si el firewall tiene que habilitar los puertos 80 y/o 443 para que la aplicación sea accesible desde el exterior, no podrá hacer nada para detectar entradas maliciosas del cliente, y por supuesto no representa una protección contra ataques internos.
- El uso de SSL es una solución suficiente - SSL simplemente cubre la petición/respuesta HTTP, en el cifrado del mensaje, dificultando la intercepción del tráfico entre cliente y servidor, pero no agrega seguridad al servidor ni evita el envío de código malicioso, posiblemente incluido en el mensaje, desde el cliente.
- Los usuarios no tienen conocimiento de las reglas de seguridad para explotar alguna vulnerabilidad – existen muchas herramientas automatizadas para llevar a cabo ataques a las aplicaciones Web sin la necesidad de tener conocimientos de las reglas aplicadas a la seguridad. Muchas de estas herramientas cuentan con una GUI y solo basta leer un tutorial en internet o ver algún video de cómo usarlas para poder realizar un ataque exitoso.
- Al agregar Pulg-ins o librerías externas estas no cuentan con defensa a vulnerabilidades – muchas fallas de seguridad se deben a este tipo de componentes desarrollados por otras empresas o desarrolladores que no han sido validados o probados adecuadamente, o contienen las firmas necesarias.

La gran mayoría de los desarrolladores suelen buscar ayuda en sitios Web que ofrecen tutoriales o códigos de ejemplo con el fin de resolver problemas comunes de codificación, pero en estos sitios a menudo enseñan prácticas de codificación inseguras y los ejemplos que suelen dar por lo general no toman en consideración las buenas prácticas de seguridad, pudiendo presentar en forma oculta deficiencias de diseño y la creación de vulnerabilidades ocultas al desarrollador (Clark, 2009).

Debido a lo comentado, la seguridad de aplicaciones Web debe abordarse a través de varios niveles en las aplicaciones y en múltiples capas del modelo OSI. Una alternativa es analizar las peticiones HTTP antes de llegar al servidor y poder clasificar o detectar la inyección SQL. Se ha desarrollado una herramienta conocida como Firewall de Aplicación Web (Web Application Firewall o WAF) para tal fin. Con este tipo de Firewall, se pueden tomar las medidas adecuadas para interrumpir las peticiones realizadas por el cliente (por ejemplo el Navegador Web), registrar eventos y advertir a los administradores de la aplicación para que estos tomen las medidas necesarias. Para poder realizarlo, el WAF debe contar con un analizador de cabeceras HTTP, las cuales provienen de las peticiones realizadas por el cliente o navegador Web y se debe contar una tasa elevada de detección de inyecciones SQL para evitar pérdidas o fuga de información.

Últimamente, se han propuesto diversos enfoques y técnicas para la detección o clasificación de inyecciones SQL en las aplicaciones Web. En el presente trabajo de investigación, se propone el uso de dos técnicas para la clasificación de inyecciones SQL, las cuales se basan en encontrar similitudes en las peticiones recibidas con respecto a dos documentos generados, los cuales uno contiene peticiones normales de una aplicación web y el otro peticiones con patrones de ataques de inyección SQL. La primera es conocida como

modelo vectorial y en combinación con el método conocido como Tri-grama.

1.3 Hipótesis

El principal objetivo de este trabajo puede ser expresado con la confirmación de la siguiente hipótesis:

"El uso del clasificador de similitud de texto basado en el modelo vectorial, así como en el método de Tri-Gramas, puede ayudar a clasificar los ataques de inyecciones SQL a las aplicaciones Web con una mayor precisión".

Para poder alcanzar una mayor precisión en la detección, el clasificador debe de evitar al menos los siguientes errores:

- Falsos Positivos. Cuando se clasifica una petición HTTP como ataque o inyección SQL pero es una petición legítima. Este error puede dejar sin servicio al usuario que intenta hacer un uso legítimo del sistema.
- Falsos Negativos. Cuando una petición lleva una inyección SQL y debe ser clasificada como ataque, pero no es detectada y se clasifica como petición legítima. Este error puede llevar a pérdidas económicas o de información valiosa.

Debido a que la seguridad al 100% no existe, nuestro clasificador debe poder alcanzar una tasa de detección cercana al 100% ante ataques realizados que sean conocidos y desconocidos.

1.4 Propuesta

En el presente trabajo, el uso del modelo vectorial se basa en la idea de que el protocolo HTTP está basado en texto, por lo que se le puede aplicar técnicas de clasificación de texto para detectar inyecciones SQL.

En nuestro enfoque se hace un análisis de la adaptación del modelo vectorial, así como del modelo basado en Tri-Gramas. Estas dos técnicas nos ayudan a obtener los tokens (los cuales se encuentran basados en Tri-gramas) y asignarle pesos a cada uno, para lo que se hace uso del Término-Frecuencia – Frecuencia Inversa del Documento (TF-IDF por sus siglas en inglés). Los tokens se obtienen de la petición HTTP. Para clasificar una petición como legítima o como ataque se hace uso de la fórmula de similitud coseno.

Estas técnicas son muy usadas para encontrar la correlación entre una consulta y distintos documentos. Los Tri-gramas nos pueden ayudar a relacionar los tokens con las consultas SQL debido a que muchas de las palabras claves y caracteres especiales son muy frecuentes y suelen aparecer juntos, tales como: punto y coma (;), UNION y SELECT, '=', etc.

Como observación, no se propondrán algoritmos nuevos para la detección de inyecciones SQL.

1.5 Objetivos

1.5.1 General

Identificar o clasificar con una alta precisión los ataques de inyecciones SQL realizados a las aplicaciones Web, a partir de los datos enviados mediante la URL y cabeceras HTTP. Para esto, se realiza un proceso conocido como tokenización (cada token es un Tri-grama), la obtención de pesos se hace mediante la técnica TF-IDF y la clasificación con la fórmula de similitud coseno.

1.5.2 Objetivos Específicos

Como objetivos específicos se tienen los siguientes:

- Explicar la peligrosidad de los ataques de inyección SQL.
- Diseñar e implementar un mecanismo para la detección de inyecciones SQL basado en el modelo vectorial y Tri-gramas.
- Seleccionar un conjunto de datos para el entrenamiento del sistema WAF.
- Definir una aplicación vulnerable para realizar las pruebas.
- Analizar herramientas para búsqueda de vulnerabilidades y ataques de inyección SQL.

1.6 Estructura de la Tesis

La estructura de la tesis es la siguiente:

- **Capítulo II. Antecedentes.** Este capítulo proporciona una introducción a las aplicaciones Web, explicando las características de su funcionamiento y su entorno de ejecución. También nos presenta una introducción a la seguridad, explicando que es la seguridad y porque es necesaria, así como los tipos de inyección SQL que se pueden llevar a cabo en las aplicaciones Web. Finalmente, nos da una introducción a las características y funcionamiento de los sistemas Firewall de Aplicación Web (WAF), así como al modelo vectorial y el método de Tri-gramas.
- **Capítulo III. Actualidad en la Detección de Inyecciones SQL.** Este capítulo nos da un repaso de los distintos trabajos relevantes relacionados con los sistemas para la detección de inyección SQL.
- **Capítulo IV. Diseño de un Clasificador de Inyecciones SQL.** Esté capítulo describe el diseño de un clasificador de similitudes de inyecciones SQL para la

detección de este tipo de ataques haciendo uso del modelo vectorial y los Tri-gramas.

- **Capítulo V. Evaluación del Clasificador.** Con el fin de comprobar la validez de este método para alcanzar una alta precisión en la detección de inyecciones SQL, se propone una serie de experimentos para: a) Ataques con las inyecciones SQL del conjunto de entrenamiento del clasificador, y b) Ataques con inyecciones SQL de un conjunto de pruebas (inyecciones no conocidas por el clasificador).
- **Capítulo VI. Conclusiones.** En este capítulo se presenta un resumen crítico del trabajo de investigación realizado, reafirmando la hipótesis y las contribuciones de la tesis. De igual forma, se presenta una sección de trabajo a futuro, que resume los próximos pasos a seguir en la investigación de ataques a aplicaciones Web.

CAPÍTULO II

Antecedentes

"Saber romper medidas de seguridad no hace que seas hacker, al igual que saber hacer una puerta en un coche no te convierte en un ingeniero automotriz"

Eric Raymond

El objetivo de este capítulo es proporcionar: (a) una introducción a las aplicaciones Web, explicando las bases de funcionalidad del protocolo HTTP, las características de su funcionamiento y su entorno de ejecución; (b) una introducción a los conceptos de seguridad y una descripción y ejemplos de los diferentes tipos de inyecciones SQL; finalmente (c) una descripción de las técnicas de recuperación de información utilizadas para la clasificación de los ataques de inyecciones SQL.

2.1 Aplicaciones Web

La World Wide Web, también llamada la Web o WWW, fue desarrollada entre marzo de 1989 y diciembre de 1990 por el inglés Tim Berners-Lee. La Web es un sistema de distribución de documentos de hipertexto o hipermedios interconectados y accesibles vía Internet (Web, 4).

Las aplicaciones Web se pueden definir de varias formas:

a) Las aplicaciones Web son programas que residen en un servidor Web para dar la funcionalidad al usuario de utilizar más de una página" (Graves, 2007).
b) "Una aplicación Web es una aplicación que es accedida vía el protocolo de transferencia de Hipertexto o HTTP" (Scambray, Liu, & Sima, 2010).

c) “Una aplicación de software, ejecutada por un servidor Web, que responde a las peticiones de páginas dinámicas a través del uso del protocolo HTTP” (WASC, 2011).

En un inicio, las aplicaciones Web eran estáticas, esto quiere decir que solo estaban conformadas de texto, enlaces e imágenes. Los navegadores eran muy simples y solo podían reproducir HTML. A este tipo de páginas también se les conoce como Web 1.0 y no eran más complejas que el modelo cliente-servidor.

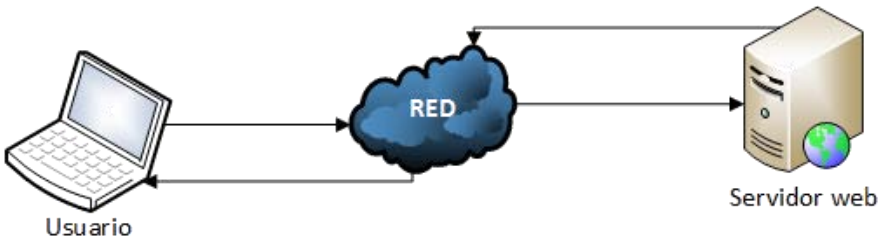

Figura 1. Modelo Cliente-Servidor

Como se puede apreciar en la Figura 1, la Web 1.0 se conformaba del cliente, en este caso el navegador Web, y un servidor que responde a las peticiones del cliente devolviendo la página HTML.

Conforme las aplicaciones comenzaron a ser más complejas y más interactivas, aunado a que cada vez los navegadores y servidores permitían más funcionalidades, se pasó a lo que se conoce como Web 2.0. Como ejemplos de aplicaciones de la Web 2.0 podemos encontrar: redes sociales, media sharing, comercio electrónico, wikis, blogs, Web mail, páginas de subastas, sistemas bancarios, etc.

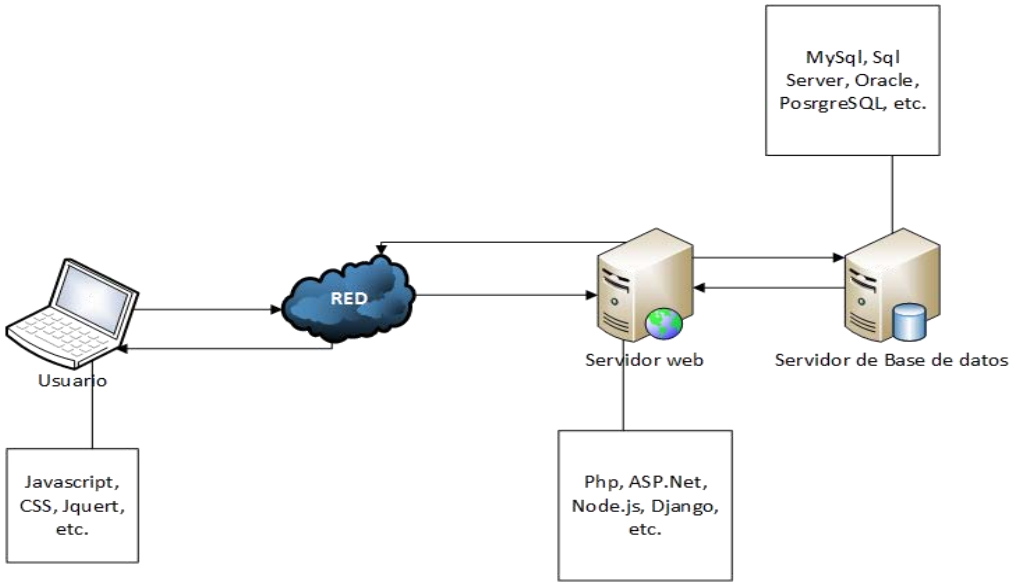

Figura 2. Arquitectura Web 2.0

Como puede apreciarse en la Figura 2, la Web 2.0 cuenta con una mejor organización para tratar con la complejidad requerida por las aplicaciones actuales. De igual forma, se pueden apreciar las nuevas tecnologías que se han ido incorporando a navegadores y servidores y la aparición de un nuevo servidor para almacenar los datos de dichas aplicaciones.

La evolución de la Web no se ha detenido, actualmente estamos entrando a la era de la Web 3.0, la Web semántica. Ésta es conocida como la Web inteligente y facilita el entendimiento de la información. Estos nuevos cambios se deben a los nuevos servicios como el de la computación en la nube, plataformas y aplicaciones móviles que hacen posible el cómputo ubicuo y los metadatos asociados a tales aplicaciones. Estos metadatos pueden ser procesados por agentes inteligentes, con algoritmos que han sido desarrollados en el campo de procesamiento de lenguaje natural, máquinas de aprendizaje y algoritmos semánticos. Como ejemplo se tienen los sistemas de recomendación que se pueden encontrar en las plataformas de Netflix y Amazon.

Aún con estos cambios, una aplicación Web básica cuenta con los siguientes componentes

descritos por Cross (Croos, y otros, 2007):

Servidor Web. Un servidor Web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador Web. Para la transmisión de todos los datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa. El servicio proporcionado a través del puerto 80 está ligado en forma estándar al protocolo HTTP, utilizando el puerto 80 para el http y el 443 para la versión segura https en conjunción con SSL.

- **Contenido de la aplicación.** El contenido de la aplicación en el entorno de ejecución, se encuentra localizado en el servidor, se comporta como un programa interactivo que maneja, controla y da respuesta a las peticiones del navegador Web a fin de realizar ciertas funciones. La forma en que los programas se ejecutan o interpretan dependen del mecanismo utilizado para la generación del código objeto y del mecanismo utilizado para su ejecución. Ejemplos de éstos pueden ser PHP, ASP.Net, Python, Ruby, Javascript, etc.
- **Almacén de Datos.** El almacén de datos es típicamente una base de datos, pero podrían ser: archivos planos, o cualquier tipo de unidad de almacenamiento en donde la aplicación pueda almacenar o recuperar datos. El almacén de datos puede

residir en un equipo completamente diferente al equipo en donde se localiza el servidor Web ya que tanto el servidor Web como el almacén de datos no necesitan estar en la misma red, sustentando su comunicación únicamente a través de una conexión de red. Los datos deben ser procesados con la máxima seguridad posible, esto se debe a que son el recurso más importante de la aplicación. Ejemplos de bases de datos pueden ser: MySQL, Oracle, SQL Server, MongoDb, etc.

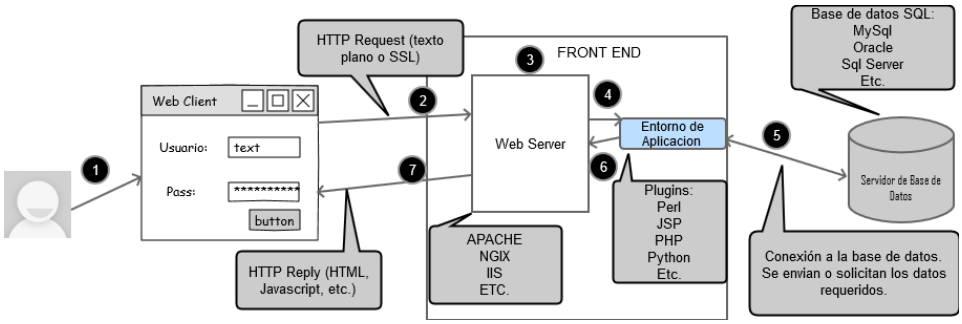

Figura 3. Arquitectura básica de los componentes de una aplicación Web.

Puede observarse en la figura 3 la existencia de la capa del OSI correspondiente al nivel de presentación o cliente. Ésta se encarga de establecer la conexión al servidor y realizar las peticiones de servicios o recursos. En este componente podemos encontrar a los navegadores como Chrome, Firefox, Internet Explorer, etc. Sin embargo, también puede ser cualquier motor de renderizado.

Como se menciona en el capítulo 1, las nuevas tecnologías incorporadas a las aplicaciones Web han ayudado en gran parte a innovar en nuevos modelos de negocio para las empresas o en sistemas que son accesibles desde cualquier parte del mundo. Sin embargo, no solo se ha atraído el interés de los empresarios o emprendedores, sino también el de los posibles atacantes con el fin de obtener beneficios de las vulnerabilidades que puedan tener estos sistemas.

2.1.1 HTTP

El Protocolo de Transferencia de Hipertexto (HTTP por sus siglas en ingles), es un protocolo a nivel de aplicación, genérico y sin estado (Fielding, y otros, 1999). El formato de hipertexto utilizado por el protocolo escrito en html, se presenta en la línea de comandos del navegador para llevar a cabo la petición o consulta de información de la siguiente manera genérica:

http://localhost:80/

O bien:

http://dirección en la Web:puerto del servicio/solicitud

La operación general descrita en el RFC 2616 es de la siguiente forma:

> Se basa en el modelo de Petición-Respuesta. El cliente envía una petición HTTP al servidor con los siguientes datos: un método para indicar la acción a realizar sobre el recurso (GET, POST), un identificador de recursos uniforme (URI por sus siglas en inglés), http:/…./ y en su contenido como encabezado la versión del protocolo, por ejemplo HTTP 1.1, y las cabeceras del mensaje de petición. El servidor envía una respuesta HTTP que contiene una línea de estado, la versión del protocolo del mensaje y un código de estado, el cual indica el éxito o si hubo un error al procesar la petición; seguido de las cabeceras de mensaje de respuesta (Fielding, y otros, 1999).

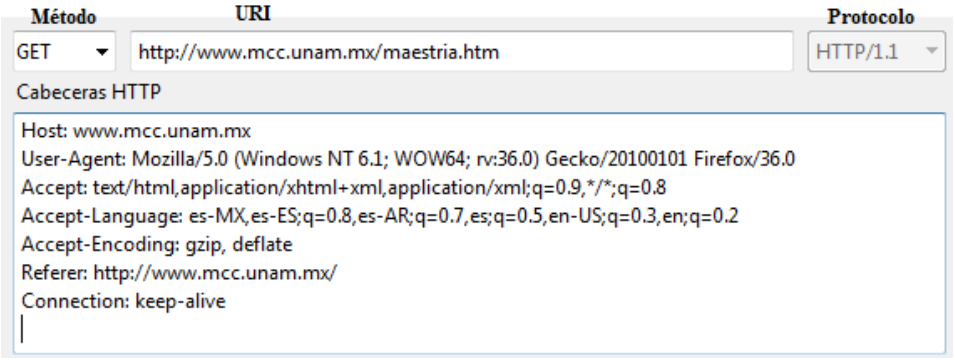

Figura 4. Petición HTTP

En la figura 4 se pueden observar los datos que han sido enviados con la petición realizada por el navegador Firefox. La primera línea de la petición identifica la acción a realizar, el recurso (URI) y el protocolo (HTTP 1.1). De igual forma, se puede enviar información en el cuerpo de la petición o como una consulta.

En el ejemplo mostrado se requiere al servidor, el cual regresa una respuesta con la página en formato html correspondiente a la maestría en ciencias de la computación de la UNAM.

La consulta es un componente de la solicitud, y en ella puede utilizarse el formato siguiente:

http:// dirección IP : #puerto del servicio/ aplicación o acción solicitada seguida de un signo de interrogación (?) Seguida de información requerida por la aplicación para efectuar la acción.

El mensaje después del carácter "?". Contiene pares de parámetro-valor, por ejemplo:

```
GET /wackopicko/users/login.php?username=ivan&password=iimas
```

Si la acción es POST los datos se incluyen en el cuerpo del mensaje como pares de parámetro-valor.

username=ivan&password=iimas

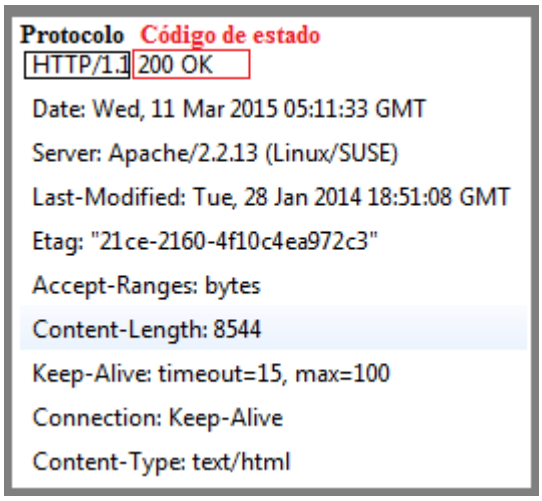

Figura 5. Respuesta HTTP enviada por el servidor.

Una vez que el servidor ha recibido y procesado la solicitud, éste debe devolver un mensaje de respuesta HTTP hacia el cliente. En la figura 5, se tienen las cabeceras que han sido devueltas como respuesta del servidor a la petición de la figura 4. La primera línea de la respuesta es conocida como la línea de estado. Se compone de la versión del protocolo HTTP que se ajusta a la respuesta, seguida por un número que representa al código de estado y su estatuto de texto explicación.

Encabezados | Respuesta | HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>MCC</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

  <!-- los estilos -->
  <LINK REL="STYLESHEET" TYPE="text/css" HREF="estilos/mcc.css">

  <!-- carga de imagenes dinámicas -->
  <script language="JavaScript" src="js/preloadImages.js"></script>

  <style type="text/css">
<!--
.style9 {font-family: Arial, Helvetica, sans-serif}
-->
  </style>
</head>
<body bgcolor="#336699" background="images/fondo.gif" link="#336699" vlink="#336699" alink="#336699"
>
```

Figura 6. Código HTML devuelto como respuesta del servidor.

En la figura 6 se observa el contenido del cuerpo de la respuesta HTTP. Se puede apreciar que contiene código en HTML y en CSS, los cuales definen como se mostrará la página solicitada.

Como se ha visto, las solicitudes HTTP del cliente incluyen en su formato la solicitud de ejecución de una acción a nivel de comando (también conocida como método o verbo). HTTP define 8 métodos, tal como puede verse en la tabla 1, que indican la acción que desea que se efectúe sobre el recurso identificado. Para más información acerca de los métodos y códigos de estado puede consultar (Fielding, y otros, 1999).

Tabla 1. Métodos HTTP.

Método	Significado
GET	Devuelve el recurso identificado en la URL pedida.
HEAD	Funciona como el GET, pero sin que el servidor devuelva el cuerpo del mensaje. Es decir, sólo se devuelve la información de cabecera.
POST	Indica al servidor que se prepare para recibir información del cliente. Suele usarse para enviar información desde formularios.
PUT	Envía el recurso identificado en la URL desde el cliente hacia el servidor.
OPTIONS	Pide información sobre las características de comunicación proporcionadas por el servidor. Le permite al cliente negociar los parámetros de comunicación.
TRACE	Inicia un ciclo de mensajes de petición. Se usa para depuración y permite al cliente ver lo que el servidor recibe en el otro lado.
DELETE	Solicita al servidor que borre el recurso identificado con el URL.
CONNECT	Este método se reserva para uso con proxys. Permitirá que un proxy pueda dinámicamente convertirse en un túnel. Por ejemplo para comunicaciones con SSL.

Para mantener las sesiones Web el navegador y el servidor Web comparten un identificador único definido a nivel de TOKEN que el navegador Web incluye en cada petición **HTTP** o

HTTPS realizada al servidor. De este modo, por medio del identificador, el servidor Web puede reconocer que la petición que recibe pertenece a una determinada sesión, reconocida por el TOKEN asignado en la primera comunicación, y le permite almacenar la información de esta petición que pueda ser de interés y responder a ella según la información almacenada anteriormente. Este mecanismo puede también estar basado en cookies.

2.2 Seguridad en Aplicaciones Web

El término seguridad proviene del latín securitas, lo cual se puede referir como la ausencia de riesgo o también a la confianza en algo o alguien. El riesgo en aplicaciones Web ha aumentado y su desarrollo puede dejar huecos de seguridad que se convierten en vulnerabilidades, debido a que cada día son más sofisticadas y cada vez más complejas técnicamente. Algunos problemas tales como la inyección SQL, a pesar de ser antiguos y muy conocidos siguen prevaleciendo.

El objetivo de la seguridad en aplicaciones Web intenta prevenir los riesgos como la pérdida de información, la no disponibilidad del servicio que ofrece y accesos a secciones no autorizadas.

Conforme las empresas y gobierno han ido mudando sus sistemas de escritorio a aplicaciones Web, éstas se han vuelto más atractivas para los atacantes. Scambray en (Scambray, Liu, & Sima, 2010) señala algunas de las características de las aplicaciones Web que las hacen tan atractivas:

- **Ubicuidad**: Las aplicaciones Web están en casi todas partes hoy y siguen extendiéndose rápidamente a través de redes públicas y privadas. Los atacantes pueden encontrar objetivos de interés en cualquier momento.

- **Técnicas Simples**. Las técnicas de ataques a las aplicaciones Web son simples y fáciles de entender, incluso para personas no técnicas, ya que son basadas en su mayoría en texto. Esto hace que la manipulación de entradas de la aplicación sea bastante trivial.
- **Anonimato**. En Internet es muy fácil de lanzar ataques que no puedan ser rastreados. Se puede tener acceso a proxies para ocultar el ataque, los cuales son abundantes y de fácil acceso en internet. Se pueden lanzar ataques a través de varios proxies, lo cual hace más sofisticado el ataque y más difícil de rastrear.
- **Pasar los Firewalls**. Las solicitudes HTTP/S son permitidas por la mayoría de las políticas típicas de los cortafuegos o firewalls. Esta configuración probablemente va a aumentar su frecuencia a medida que más y más aplicaciones migren a HTTP.
- **Código Personalizado**. Con la proliferación de plataformas de desarrollo Web de fácil acceso como ASP.NET y LAMP (Linux / Apache / MySQL / PHP), la mayoría de las aplicaciones Web son ensambladas por los desarrolladores que tienen poca experiencia.
- **Seguridad Inmadura**. HTTP por sí mismo no implementa sesiones para separar a usuarios únicos. La autenticación y autorización a nivel sesión se consideró años después de que la tecnología comenzó a ser popular y aún sigue evolucionando en nuestros días. Muchos desarrolladores codifican sus manejadores de sesiones y autenticaciones pero muchas veces se equivocan o pierden el control ya en la operación.
- **Cambio Constante**. Por lo general muchas personas "tocan" la aplicación Web: desarrolladores, administradores de sistemas, y gestores de contenido de todo tipo (se ha visto que muchas empresas le dan acceso directo a las aplicaciones en

producción al equipo de marketing). Muy pocas de estas personas tienen la formación adecuada en seguridad y sin embargo, tienen el poder de hacer cambios en una aplicación compleja.

- **Dinero**. A pesar de los contratiempos de la era punto-com, está claro que el comercio electrónico a través de HTTP apoya a muchos negocios lucrativos. No es sorprendente que las estadísticas recientes indican que la motivación para atacar o vulnerar una aplicación Web se ha movido de ser famoso a tener una fortuna, en paralelo con la maduración de la propia Web.

Conforme los usuarios de las aplicaciones se han ido adaptando a su uso, han hecho conciencia de que la seguridad es un problema a solucionar para las aplicaciones Web. Stuttard (Stuttard & Pinto, 2011) menciona lo siguiente:

Si se consulta la página de preguntas frecuentes de una aplicación típica, ¿se puede estar seguro de que en realidad es "seguro"?

Algunos de los ejemplos que pueden encontrarse en las preguntas frecuentes son:

"Este sitio es absolutamente seguro. Se ha diseñado para utilizar la tecnología Secure Socket Layer (SSL) de 128-bit para evitar que usuarios no autorizados puedan ver su información. Usted puede usar este sitio con la tranquilidad de que sus datos están seguros con nosotros" (Stuttard & Pinto, 2011).

"Nos tomamos muy en serio la seguridad. Nuestro sitio Web se escanea diariamente para asegurar que seguimos siendo compatible con PCI y a salvo de los hackers. Se puede ver la fecha de la última exploración en el logo de abajo, y se le garantiza que nuestro sitio Web es seguro de usar" (Stuttard & Pinto, 2011).

A pesar de que todo esto ayuda en parte a mantener una mayor seguridad, las aplicaciones

Web siguen siendo inseguras y cada vez han ido aumentando los ataques, tal como se menciona en el primer capítulo del presente trabajo.

FISMA[1] y FIPS[2] definen 3 objetivos de seguridad para que un sistema sea fiable o seguro (NIST, 2008):

Tabla 2. Objetivos de la seguridad de la información y sistemas de información, también conocidos como CIA.

Objetivos	Definición de FISMA	Definición de FIPS
Confidencialidad	"Preservar restricciones autorizadas sobre el acceso y la divulgación de la información, incluidos los medios para la protección de la intimidad personal y propiedad de la información ..."	Una pérdida de confidencialidad es la divulgación no autorizada de información.
Integridad	"Protección contra la modificación de información incorrecta o destrucción, e incluye asegurar el no repudio de la información y autenticidad ..."	Una pérdida de la integridad es la modificación o destrucción de información no autorizada.
Disponibilidad	"Garantizar el acceso oportuno y confiable y uso de información ..."	Una pérdida de disponibilidad es la interrupción del acceso o al uso de la información de un sistema de información.

Uno de los objetivos de un sistema seguro es preservar su confidencialidad. Los datos

[1] Federal Information Security Management Act
[2] Federal Information Processing Standard

deben ser revelados solo a usuarios autorizados. La criptografía y la autorización y control de acceso se utilizan para ocultar el contenido de los datos. La autorización define el acceso permitido a los recursos y el control de acceso es una manera de hacer cumplir la autorización. Otro de los objetivos de la seguridad es el preservar su integridad, es decir, evitar inconsistencias o datos erróneos. La criptografía, autorización y control de acceso, así como los mecanismos de identificación y anti-replay se utilizan para hacer cumplir este objetivo de seguridad. Y como tercer objetivo se debe preservar la disponibilidad. Esto se refiere a la capacidad de los usuarios legítimos de utilizar la información o el recurso deseado cuando sea necesario.

La seguridad en aplicaciones Web, y en general la seguridad, dependen tanto de políticas como de mecanismos que se deben cumplir y hacer cumplir. Muchas veces los ataques contra la seguridad vienen por descuidos de estas políticas y mecanismos por parte de los empleados, desarrolladores o de los mismos administradores del servidor. En la siguiente sección se da una breve explicación de cómo suele llevarse a cabo un ataque de inyección SQL y los resultados en caso de que éste se lleve con éxito.

2.3 Fallas de Inyección

OWASP menciona en (OWASP, 2013) que las fallas de inyección ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete y ejecutar comandos no intencionados o acceder a datos no autorizados. El atacante realiza la inyección del comando o consulta en los campos de entrada de datos, tales como el de contraseñas o comentarios. De igual forma, puede ser directamente a los parámetros de la consulta o cuerpo de la petición HTTP, o en

otras cabeceras como el de las cookies. Los ataques no solo se pueden dirigir a las aplicaciones Web, sino también a cualquier otro tipo de software que sea vulnerable a los ataques de inyección. Estas vulnerabilidades muchas veces se deben a la falta de validación de las entradas de formularios o de los parámetros que se reciben en el servidor. Los ataques más conocidos dentro de los ataques de inyección son la inyección SQL, Inyección de Comandos, Inyecciones LDAP e Inyección NoSQL.

Los ataques de inyección son fáciles de llevar a cabo mediante herramientas especializadas. Esto hace que los atacantes no necesariamente deban ser expertos o tener experiencia para poder llevar a cabo un ataque a una aplicación Web, tal como era en el pasado. Una persona que se inicia en la informática puede llevar a cabo un ataque en cuestión de minutos y obtener información importante de alguna empresa, persona o gobierno. Esto se debe, a que se pueden encontrar herramientas automatizadas, gratuitas y fáciles de usar (incluso ya cuentan con interfaz gráfica).

La finalidad de estos ataques puede ser la de obtener datos privados, acceso ilimitado, redirigir a los usuarios a otras aplicaciones Web para descarga de malware o phishing, robar las sesiones o credenciales de los usuarios del sistema, etc. Incluso, existen páginas Web en donde se venden datos de tarjetas de crédito, los cuales han sido obtenidos de bases de datos de alguna aplicación de comercio electrónico por medio de una inyección SQL.

2.3.1 Inyecciones SQL

Como se ha mencionado, uno de los ataques más frecuentes a las aplicaciones Web es el de las Inyecciones SQL. El consorcio de seguridad en aplicaciones Web (WASC por sus siglas en inglés) define a las inyecciones SQL como (WASC, 2010):

Una técnica de ataque usada para explotar aplicaciones que construyen sentencias SQL mediante entradas proporcionadas por el usuario. Cuando tienen éxito, el atacante es capaz de cambiar la lógica de las sentencias SQL ejecutadas en la base de datos.

Por su parte OWASP la define de la siguiente manera (OWASP, 2012):

Un ataque por inyección SQL consiste en la inserción o "inyección" de una consulta SQL por medio de los datos de entrada desde el cliente hacia la aplicación. Un ataque por inyección SQL exitoso puede leer información sensible desde la base de datos, modificar la información (Insert/ Update/ Delete), ejecutar operaciones de administración sobre la base de datos (tal como parar la base de datos), recuperar el contenido de un determinado archivo presente sobre el sistema de archivos del DBMS y en algunos casos emitir comandos al sistema operativo.

La inyección SQL se debe a la falta de validaciones y/o filtrado de las variables y el uso adecuado de las herramientas que proporcionan los lenguajes y scripts. En (WASC, 2010), se identifica que en las fases de diseño e implementación, del ciclo de vida de desarrollo de una aplicación, es donde suelen introducirse las vulnerabilidades que pueden ser explotadas mediante inyecciones SQL.

Las inyecciones SQL son una de las vulnerabilidades más conocidas y peligrosas. No es una vulnerabilidad que afecte directamente a la aplicación Web, sin embargo, ésta es un medio para poder llevar a cabo el ataque. En su libro, Clarke (Clark, 2009) afirma:

La inyección SQL es una de las vulnerabilidades más devastadoras para impactar a un negocio, ya que puede conducir a la exposición de toda la información sensible almacenada en la base de datos de una aplicación, tales como nombres de usuario, contraseñas, nombres,

direcciones, números de teléfono, y números de tarjetas de crédito.

Esto ha sido comprobado por muchas empresas importantes en el área de tecnologías, tales como SONY, Microsoft, PayPal, Yahoo! y Apple, que han sufrido este tipo de ataques en los últimos años.

De los objetivos de seguridad vistos en el apartado 2.2 del presente trabajo, la inyección SQL puede traer las siguientes consecuencias (OWASP, 2012):

- **La confidencialidad puede verse afectada**. Dado que las bases de datos SQL generalmente almacenan información valiosa o sensible, la pérdida de la confiabilidad es un problema frecuente con este tipo de vulnerabilidad.
- **La autenticación**. Si se utilizan consultas SQL para verificar nombres de usuarios y contraseñas y no se tienen las correctas validaciones, puede ser posible conectarse a un sistema como otro usuario sin conocimiento previo de la contraseña.
- **Autorización**. Si la información de autorización es almacenada en una base de datos SQL, puede ser posible cambiar esa información mediante la explotación exitosa de una vulnerabilidad por inyección SQL.
- **Integridad**. Así como puede ser posible leer información sensible, también es posible realizar cambios o incluso borrar esta información mediante un ataque de inyección SQL.

En la guía de pruebas de OWASP (OWASP, 2008), los tipos de ataques de inyección SQL están clasificados de la siguiente forma:

- **Inband**. Los datos son extraídos usando el mismo canal que se usa para inyectar el código SQL. Este es el tipo de ataque más simple, en el que los datos recibidos se muestran en la propia aplicación Web.

- **Out of Band**. Los datos son extraídos usando un canal diferente (por ejemplo, un correo que se genera con el resultado de la consulta y se envía al auditor o atacante).
- **Inference**. No hay transferencia de datos, pero el auditor o atacante puede reconstruir la información enviando peticiones y observando el comportamiento que mantiene el servidor de bases de datos. Este ataque también es conocido como inyección SQL ciega (Blind SQL Injection).

Una de las formas más simples de verificar si una aplicación es vulnerable a una inyección SQL, es añadiendo una comilla simple (') o un punto y coma al campo que se requiere probar. También pueden usarse palabras reservadas como AND, OR o comentarios (--) del lenguaje SQL para intentar modificar la consulta original de la aplicación. Si el campo no está filtrado correctamente, la base de datos genera un mensaje de error que puede ser visible en la aplicación. Los errores pueden ser útiles, debido a que ofrecen una gran cantidad de información que pueden llevar al atacante a crear una inyección con éxito (OWASP, 2008).

Un ejemplo de error, que genera la base de datos SQL Server de Microsoft, al intentar inyectar una comilla simple " ' " en un parámetro de la URL, es el siguiente:

URL normal:

www.ejemplo.com/productos.aspx?categoría=computadoras

URLcon la inyección al final de la palabra ataque:

www.ejemplo.com/productos.aspx?categoría=ataque'

Error generado a causa de la comilla:

```
Server Error in '/' Application. Unclosed quotation mark before the character string ataque;'.Description: An unhandled exception occurred during the execution of the current Web request. Please review the stack trace for more information about the error and where it originated in the code. Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark before the character string ataque;'.
```

El error se produce debido a que la comilla genera que la consulta no este con el formato apropiado para poder ejecutarse en la base de datos. De igual forma, este error puede indicarle al atacante o a la herramienta que la aplicación puede ser vulnerable a inyección SQL, ya que no se está filtrando el uso de este tipo de caracteres. Al inyectar la comilla, la consulta queda de la siguiente forma:

```
SELECT * FROM productos WHERE categoria='ataque''
```

Otra forma de comprobar si una aplicación es vulnerable, es la de inyectar cadenas en parámetros que aceptan números, como por ejemplo:

URL normal:

```
www.ejemplo.com/productos.aspx?idProducto=67
```

URL con una cadena inyectada en el parámetro idProducto:

```
www.ejemplo.com/productos.aspx?idProducto=ataque
```

Error generado a causa de la cadena "ataque":

```
  Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
  [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting
the
  varchar value 'ataque' to a column of data type int.
  /productos.aspx, line 113
```

Otro tipo de ataque muy común es el conocido como de tautologías. En este tipo, los atacantes inyectan algún o algunos tokens SQL en la entrada del usuario, con lo cual se logra que la cláusula de selección de una consulta SQL sea verdadera (Moosa, 2010). Este tipo de ataque suele aplicarse a los formularios de inicio de sesión para acceder sin contar con las credenciales registradas.

Figura 7. Ataque de inyección SQL para iniciar sesión.

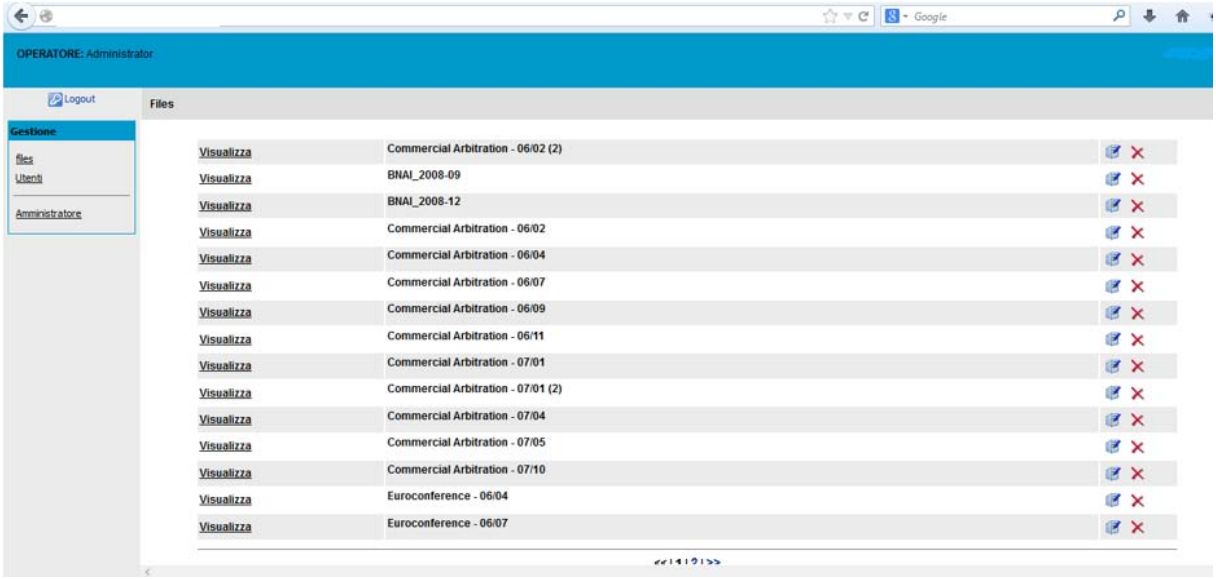

Figura 8. Panel de administración del usuario "*Administrator*".

Como puede apreciarse en la figura 7, se ha realizado la inyección SQL para que la consulta generada en la aplicación se haga verdadera, esto se debe a que 1 siempre es igual a 1.

En la figura 8, se puede apreciar que con la inyección se tiene acceso al panel de administrador con el usuario Administrator sin haber proporcionado las credenciales de éste. Después de inyectar los tokens mencionados, la consulta posiblemente se esté realizando de la siguiente forma:

```
SELECT * FROM Users WHERE user = '' or '1'='1' AND password =
'' or '1'='1';
```

Como se ha comentado, se cuenta con un tipo de ataque de inyección SQL conocido como Inyección SQL Ciega. Este tipo de ataque a menudo se basa en diferentes tiempos de respuesta del servidor Web para descubrir otra información sobre la base de datos (Moosa, 2010). Este ataque se lleva a cabo, debido a que en algunas aplicaciones los mensajes de error no se muestran al usuario o los desarrolladores suelen capturar los errores y mostrar una página personalizada de error. En esta situación el atacante debe usar ingeniería inversa para conseguir la lógica de la consulta original.

Para saber si una aplicación es vulnerable a la inyección SQL Ciega existen varias técnicas, pero las más antiguas y que aún suelen usarse son los ataques de tiempo. Esto se debe a que SQL puede ejecutar comandos del sistema operativo, por ejemplo: si la base de datos es SQL Server se hace uso de WAITFOR DELAY; Si es MYSQL puede hacerse uso de SLEEP().

En algunas ocasiones los desarrolladores suelen realizar el escape de caracteres especiales (como comillas, guiones, etc.) o de algunas palabras claves que puede haber en el lenguaje de consulta SQL. Pero esto no es suficiente para evitar las inyecciones y un método que suelen usar los atacantes es el de codificar o cifrar la inyección. Esto es gracias a que los navegadores aceptan que algunos símbolos y caracteres estén en codificación URL o hexadecimal para que puedan interpretarlos correctamente. A continuación se tiene un ejemplo de codificación URL:

```
www.ejemplo.com%2Fproductos.aspx%3FidProducto%3D%27%20or%20%27
1%27%20%3D%20%271
```

URL decodificada:

```
www.ejemplo.com/productos.aspx?idProducto=' or '1' = '1
```

Como se ha podido observar los ataques de inyección SQL pueden ser simples, se necesita conocer el lenguaje de consulta SQL y tener mucho tiempo para estar generando e inyectando diferentes consultas. Actualmente se han desarrollado herramientas para automatizar y realizar los ataques (desde los más básicos a los más avanzados) de una forma más rápida y fácil. Estas herramientas son creadas para apoyar a las empresas de desarrollo o consultores de seguridad, pero de igual forma suelen ser usadas por los atacantes (muchas de ellas son gratuitas y vienen en distintas distribuciones de Linux).

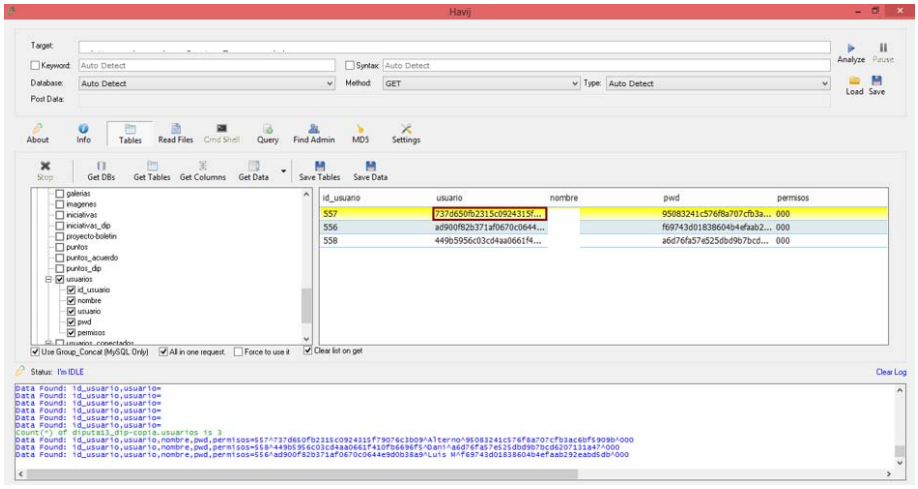

Figura 9. Ataque realizado a la página de diputados del PRD.

En la figura 9, se observa la pantalla de la herramienta Havij. Con esta herramienta se obtuvieron datos de las diferentes tablas de la base de datos en cuestión de minutos. En la figura 10 se puede observar un ataque realizado con la misma herramienta a la página Web de la biblioteca del IIMAS.

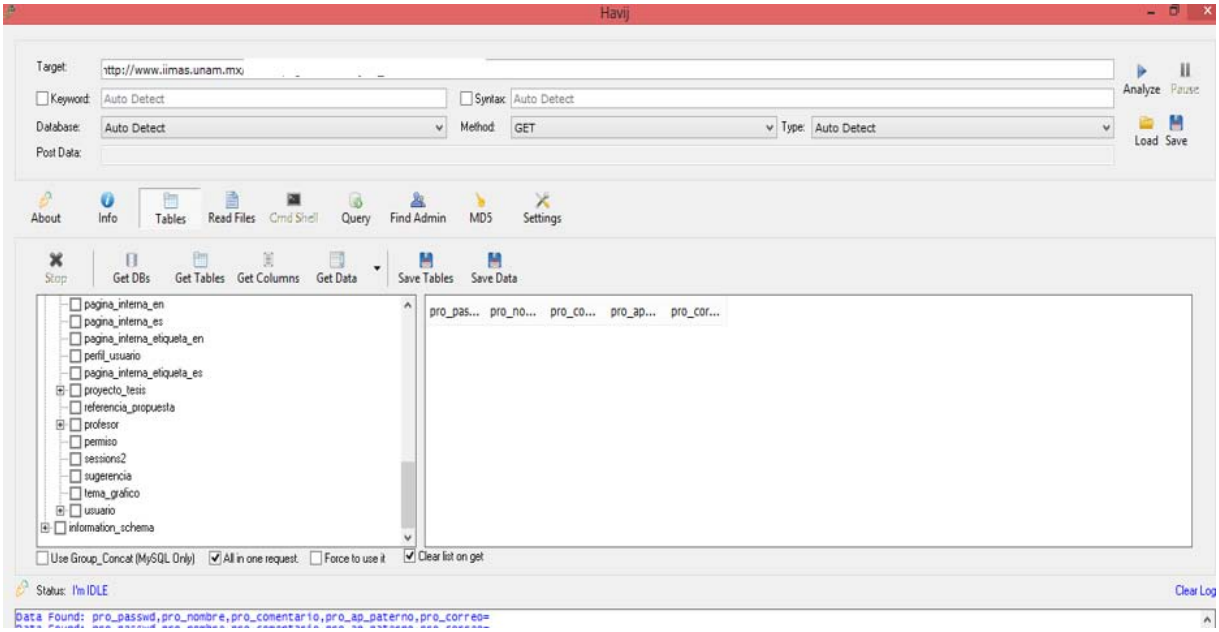

Figura 10. Ataque realizado a la página Web de la biblioteca del IIMAS.

Muchas de las técnicas que se han mostrado son usadas por los atacantes para obtener la información de la base de datos, para pasar validaciones (como en el caso de inicios de sesión), disminuir precios a los productos, etc. De igual forma, se puede apreciar que con las herramientas casi cualquier persona puede llevar a cabo un ataque, lo cual se debe a la facilidad que brindan las herramientas. Para evitar los ataques a las aplicaciones Web se han desarrollado herramientas como el firewall de aplicación Web y el firewall de bases de datos. En la siguiente sección se habla más a fondo del firewall de aplicación Web, el cual ha ganado mucha importancia como complemento para evitar los diferentes ataques que suelen realizarse a las aplicaciones Web.

2.4 Firewall de Aplicación Web (WAF)

Un firewall de aplicación Web o también conocido como WAF (Web Application Firewall) trabaja en la capa de aplicación y puede ser software o un dispositivo hardware. De acuerdo a

WASC (WASC, 2011), un WAF se define como:

> Un dispositivo intermediario, instalado entre un cliente Web y un servidor Web (ver figura 11), que analiza los mensajes de la capa 7 del modelo OSI, en busca de violaciones de las políticas de seguridad programadas. Un firewall de aplicaciones Web se utiliza como un dispositivo de seguridad que protege al servidor Web de los ataques.

En (Gupta & Saikia, 2007) podemos encontrar la siguiente definición:

> Un Firewall de Aplicación Web es un módulo de seguridad en un dispositivo proxy de aplicación que protege al servidor de aplicaciones Web de varios ataques.

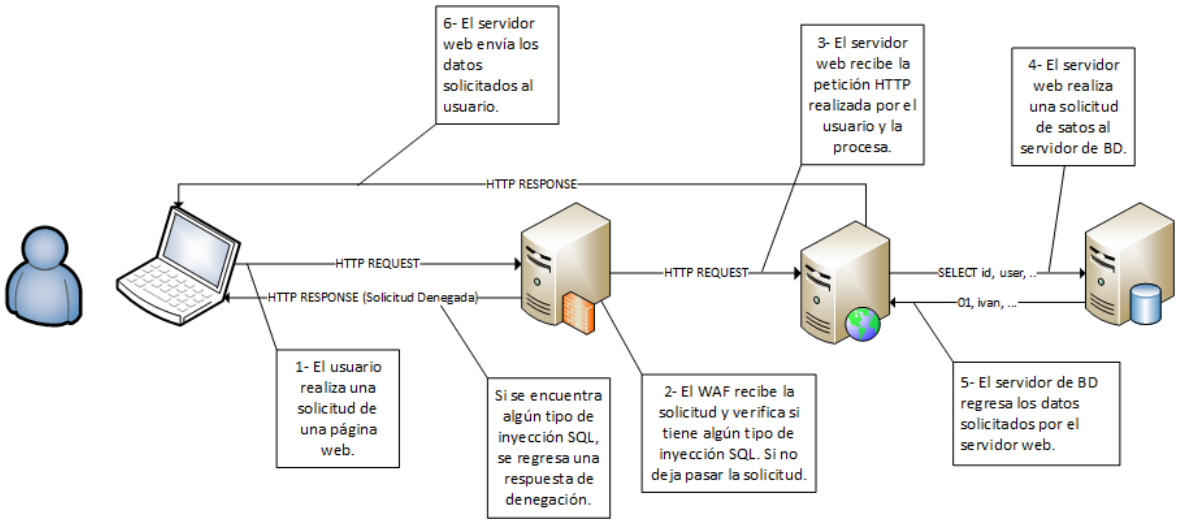

Figura 11. Arquitectura de una aplicación Web protegida por un WAF.

Tal como mencionan las definiciones anteriores, el WAF se encarga de analizar el tráfico HTTP/HTTPS dirigido a las aplicaciones Web, con el fin de protegerlas de ataques como inyección SQL, cross site scripting, manipulación de parámetros, etc. Esta última característica los diferencia de los firewalls convencionales y se debe a que los firewalls o corta fuegos convencionales suelen trabajar en la capa de red. Al ser herramientas

especializadas para la protección de aplicaciones Web y trabajar en la capa de aplicación, pueden tener un mayor control sobre las aplicaciones, a diferencia de otras herramientas como los sistemas de detección de intrusos (también conocidos como IDS) y los sistemas de prevención de intrusos (conocidos como IPS). La siguiente sección presenta y describe los modos de seguridad y los modos de operación en los que puede trabajar.

2.4.1 Arquitectura de un WAF

Como se menciona en el apartado anterior, el WAF se coloca, por lo general, detrás de un firewall de red y delante de un servidor Web. El despliegue o puesta en funcionamiento de éstos dispositivos se hacen a menudo “in-line”, esto quiere decir que todo el tráfico fluye a través del firewall de aplicación Web (Beechey, 2009). Sin embargo, algunas soluciones pueden ser "out of band" con el uso de un puerto de monitorización de red (Beechey, 2009).

Existen dos modos de seguridad, por lo que un WAF puede ser desarrollado para trabajar con alguno de los dos modos de seguridad existentes o con ambos. El modo de seguridad positivo (también conocido como lista blanca) es en el que se definen las políticas de lo que está permitido y se rechaza todo lo demás. Por ejemplo, se establecen las secciones públicas de la aplicación Web, y si el usuario intenta entrar a alguna sección del administrador o privada, el WAF lo evitaría e informaría al administrador de la aplicación. De igual forma, existe el modo de seguridad negativo (también conocido como lista negra) en el que se definen las políticas de lo que no está permitido y permite todo lo demás.

Modos de Operación

Un WAF puede tener distintos modos de operación, esto depende de cómo haya sido

desarrollado. Beechey (Beechey, 2009) en su trabajo define los siguientes modos de operación:

- **Reverse Proxy**. Es el más común, su funcionamiento es "in-line". El WAF realiza las peticiones al servidor Web en nombre del navegador o cliente de origen. Esto nos obliga a asignarle una IP y a hacer cambios en las tablas de DNS o NAT de los firewalls, de manera que el tráfico que antes iba a las granjas de los servidores Web directamente, ahora se direccionará a nuestro nuevo dispositivo. En este modo de operación se pueden soportar las siguientes características: caching, compresión, aceleración SSL, balanceo de carga y pooling de conexiones. La desventaja de un modo de proxy inverso es que puede aumentar la latencia y podría crear problemas para las aplicaciones menos tolerantes.
- **Transparent Proxy**. Su funcionamiento es similar al anterior pero en este caso no tiene una dirección IP. Permiten interactuar en modo transparente. De esta forma no es necesario realizar cambios en la topología de red ni en el flujo del tráfico. Su desventaja respecto a los basados en proxy inverso es que requieren una parada de servicio mucho mayor en los despliegues.
- **Layer 2 Bridge**. El WAF funciona como un switch de capa 2. Proporciona alto rendimiento y no implica cambios en la red, sin embargo no aporta servicios avanzados como lo hacen otros modos.
- **Monitor de Red / Out of Band**. El WAF inspecciona el tráfico de red a través de un sniffer monitorizando un puerto. Se puede configurar para que además bloquee cierto tráfico enviando reinicios TCP para interrumpir el mismo.
- **Host/Server Based**. Es software que se instala en los propios servidores Web. En

este caso se elimina cualquier problema de red adicional que pueda existir, pero hay que tener cuidado con su instalación en servidores Web que tengan mucha carga ya de por sí.

Esta herramienta si no se encuentran configurada adecuadamente puede detectar muchos falsos positivos, por lo tanto, en vez de ayudar pueden perjudicar a la disponibilidad de la aplicación. A su vez, pueden introducir un cierto retardo en las transferencias. Para atenuar este último factor, se pueden implementar distintas estrategias como aceleradores SSL, cachés, la comprensión de datos HTML, CSS y JS al enviarlos al navegador, etc. Debe considerarse esta herramienta como una capa más de seguridad. OWASP menciona los siguientes criterios a considerar para el uso de un WAF (OWASP German Chapter, 2008):

- Importancia de la aplicación(es) Web para el éxito de la organización (volumen de ventas, la reputación).
- Importancia en la pérdida de datos de la aplicación Web (datos de los clientes que se manejan, información confidencial de la empresa, reputación, etc).
- Número de aplicaciones Web.
- Condiciones legales o estándares industriales que obliguen a mantener la seguridad.
- Complejidad.
- Desempeño
- Costos de Operación.
- Escalabilidad.

2.5 Modelo de Espacio Vectorial

El modelo de espacio vectorial hace uso de los vectores para representar a la información, en

lenguaje natural, que se encuentra en los documentos. El espacio vectorial puede estar constituido por n-dimensiones, las cuales son representadas por sus términos. Los documentos que contengan más términos similares están a muy poca distancia entre si sobre el espacio. Generalmente, se hace uso de este modelo para el cálculo de relevancia de una consulta, la cual es tratada como un documento más, con otros documentos y para clasificación o agrupamiento de documentos (Manning, Raghavan , & Schütze, 2008).

Cada documento del espacio vectorial, el cual es representado por $\boldsymbol{D_i}$, contiene términos indexados, representados por $\boldsymbol{T_j}$, los términos se suelen ponderar de acuerdo a su importancia (Salton, Wong, & Yang, 1975).

$$d_i = (t_{i1}, t_{i2}, t_{i3}, \dots , t_{ij})$$

Donde, t_{ij} representa el peso de un término j.

Este modelo es muy eficiente para las aplicaciones de recuperación de documentos en el área de recuperación de información (IR por sus siglas en ingles). Sin embargo, se han realizado algunos trabajos en el campo de la seguridad Web, con buenos resultados, en los que se hace uso de este modelo para la clasificación de ataques en aplicaciones Web (Gallagher & Eliassi-Rad, 2009) y (Ulmer & Gokhale, 2010), detección de anomalías (Liao & Vemuri, 2002), así como para la detección de plagio de código fuente en el desarrollo de programas (Picazo, Villatoro, Luna, & Jaimez, 2014).

En las siguientes secciones se introducen los cálculos fundamentales para calcular los pesos y el coeficiente de similitud entre los documentos, que corresponden al grado de similaridad entre los términos y sus pesos, los cuales serán muy útiles en el desarrollo de este trabajo.

2.5.1 Frecuencia de Término – Frecuencia Inversa del Documento

Como se menciona en el apartado anterior, los términos en el modelo de espacio vectorial deben ser ponderados. Para esto, uno de los factores de ponderación más utilizados es el conocido como Frecuencia de Término - Frecuencia Inversa del Documento (TF-IDF por sus siglas en inglés).

Frecuencia de Término (TF)

La obtención de pesos de cada término del documento se puede realizar de varias formas, pero la más común es calcular el número de ocurrencias del término *t* en el documento *d*. Podríamos representarlo de la siguiente forma:

$$tf_{t,d} = f(t,d) \qquad (1)$$

En donde f(t,d), es la función que calcula las ocurrencias del término.

Tal como menciona Manning (Manning, Raghavan , & Schütze, 2008):

> *"Parece poco probable que veinte apariciones de un término en un documento verdaderamente llevan veinte veces el significado de una sola aparición".*

Debido a esto, se han realizado muchas investigaciones para encontrar variantes de las medidas *tf*. Una de las más comunes es la frecuencia escalar. En ella se hace uso del logaritmo si la frecuencia del término es mayor a 0 y en otro caso se asigna 0 (Manning, Raghavan , & Schütze, 2008).

$$wf_{t,d} = \begin{cases} 1 + \log(f(t,d), & si\ f(t,d) > 0 \\ 0, & en\ otro\ caso \end{cases} \qquad (1.1)$$

Con esta técnica, no se considera la longitud del documento, pero se reduce la importancia de un término en las colecciones con documentos de longitudes variables (Greengrass, 2000).

Esta técnica, llamada "frecuencia logarítmica," no toma explícitamente la longitud del documento o la frecuencia máxima pero sí "reduce la importancia de la frecuencia en colecciones que puedan tener longitudes muy variables de texto". También reduce el efecto de un término con una frecuencia inusualmente alta dentro de un documento dado.

Otra variante que suele usarse, conocida como máxima normalización, es la de dividir el número de ocurrencias del término t en el documento t, entre la frecuencia máxima de algún termino del documento d (Tf-idf, 2014), esto se realiza para generar un factor que varíe entre 0 y 1.

$$tf_{t,d} = \frac{f(t,d)}{\max\{f(v,d):v\in d\}} \qquad (1.2)$$

Frecuencia Inversa del Documento

La frecuencia inversa del documento (IDF por sus siglas en inglés) es una medida de cómo se distribuye el término en la colección de documentos (Greengrass, 2000). Esta medida nos ayuda a saber que tan relevante es un documento o si no lo es, con nuestra consulta. Por ejemplo, si un término aparece en todos los documentos el idf es cero.

IDF es una medida de cómo se distribuye ampliamente el término sobre la colección dada, y por lo tanto, de la probabilidad de que el término pueda ocurrir dentro de cualquier documento dado por casualidad.

IDF puede ser obtenido de la siguiente forma:

$$idf(t,D) = \log\left(\frac{|D|}{d\in D:t\in d}\right) (2)$$

|D|: es la cardinalidad de D o número de documentos en la colección.

$|\{d \in D : t \in d\}|$: es el número de documentos en donde aparece el término *t*. Debido a que puede dar 0 es común sumarle 1.

TF*IDF

Para obtener esta medida se deben multiplicar las frecuencias del término con la frecuencia inversa del documento, ver fórmula 3. En palabras de Manning (Manning, Raghavan , & Schütze, 2008), esta medida asigna un peso a cada término que es:

- Más alto cuando el término ocurre muchas veces en un pequeño número de documentos;
- Más baja cuando el término se produce un menor número de veces en un documento, o se produce en muchos documentos (ofreciendo así una señal de relevancia menos pronunciada);
- Más baja cuando se produce el término en prácticamente todos los documentos.

$$tf - idf = tf * idf \qquad (3)$$

2.5.2 Similitud Coseno

Debido a que el modelo vectorial nos ofrece la posibilidad de determinar el ángulo, que forman los vectores de los documentos en la colección con la consulta, se puede hacer uso de la fórmula conocida como medida cosenoidal.

Para obtener el coeficiente de similitud (SC por sus siglas en inglés), el cual nos dice cuán parecidos son los documentos a la consulta, se debe realizar el cálculo de la semejanza del vector consulta (d_c) y los vectores que representan los documentos de la colección (Blázquez, 2013).

$$SC(\vec{d}_c, \vec{d}_i) = \frac{\sum_{k=1}^{t} w_{ck} w_{ik}}{\sum_{k=1}^{t} (w_{ik})^2 \sum_{k=1}^{t} (w_{ck})^2} \quad (4)$$

Donde k va de 1 al total de términos de Tj, w_{ck}indica el peso del término k en el

documento $\vec{d}_c$ mientras que w_{ik} el peso del término k en el documento $\vec{d}_i$.

2.6 N-Gramas

Un N-grama es una subsecuencia de n elementos consecutivos en una secuencia dada (Choi, Kim, Choi, & Kim, 2011). Los N-gramas han sido aplicados en muchas áreas como procesamiento de lenguaje natural, recuperación de información y hasta para la detección de malware. Cuando n es igual a dos, se le conoce como bi-gramas y si n es igual a 3, como tri-gramas. En el presente trabajo se toman las palabras que conforman las peticiones HTTP como los elementos. Se trabaja con trigramas y cada uno de ellos lo llamamos token.

A continuación podemos ver un ejemplo de tri-grama:

Dado el siguiente texto "Inyecciones SQL en aplicaciones Web", si se toman como elementos a las palabras que componen la frase, sus trigramas serian:

"Inyecciones SQL en", "SQL en aplicaciones", "en aplicaciones Web"

Este modelo nos es útil al clasificar las inyecciones SQL, con el podemos evitar los falsos positivos que suelen darse en algunos métodos de clasificación, tal como se verá en los siguientes capítulos.

CAPÍTULO III

Estado del Arte

"El único sistema seguro es aquel que está apagado y desconectado, enterrado en un refugio de cemento, rodeado por gas venenoso y custodiado por guardianes bien pagados y muy bien armados. Aun así, yo no apostaría mi vida por él"

Eugene Spafford

Este capítulo presenta el trabajo relacionado a la detección de inyecciones SQL. Sin embargo, es importante mencionar que su desarrollo se ha llevado a cabo entre septiembre de 2012 a julio de 2014. Por consiguiente, se consideran solo los trabajos relacionados relevantes que han sido publicados antes de 2014.

3.1 Introducción

Desde 1998, año en el que fue mencionada por primera vez la vulnerabilidad conocida como inyección SQL, se han desarrollado muchas herramientas y técnicas para explotar este tipo de vulnerabilidad, así como para la detección de los ataques.

Las herramientas desarrolladas para evitar los ataques suelen tener dos enfoques:

- **Escáner de vulnerabilidades**. Esta herramienta cuenta con un crawler para obtener toda la información de la aplicación Web, como los campos de entrada, parámetros enviados al backend, cookies, etc. Posteriormente hace uso de técnicas y diversos algoritmos para simular la generación de ataques, y a su vez poder analizar las respuestas HTTP en busca de algún patrón de conducta tanto en los errores como en las respuestas, que pueda indicar que la aplicación es realmente vulnerable.

- **Firewall de Aplicación Web**. Como ya se ha mencionado en el capítulo 2, el WAF se encarga de analizar el tráfico HTTP/HTTPS dirigido a las aplicaciones Web, haciendo uso de diversos métodos y algoritmos de clasificación o detección, con el fin de proteger a las aplicaciones Web.

Las técnicas para la clasificación de ataques que se han implementado en los WAF son de diversos tipos. Por ejemplo, se ha hecho uso de expresiones regulares, minería de datos, agentes inteligentes, redes neuronales, etc. A continuación se mencionan algunas contribuciones que se han hecho para solucionar o evitar la inyección SQL. Se hace un especial énfasis en las técnicas de clasificación que se han propuesto en los últimos años.

3.2 Actualidad en la Detección de Inyecciones SQL

Torrano y otros (2009) presentan un firewall de aplicación Web basado en la detección de anomalías, el cual puede trabajar como IDS o como proxy. Al tener un enfoque basado en anomalías los autores afirman que puede detectar tanto ataques conocidos como desconocidos. La idea principal de éste método es la configuración de un archivo XML con el comportamiento normal de la aplicación Web, el cual ayuda a clasificar las solicitudes de entrada que se reciben. En el archivo XML se configura el comportamiento normal de la aplicación (modelo de seguridad positivo), esto quiere decir que se agregan los verbos o métodos HTTP, las cabeceras HTTP y los accesos a diferentes recursos como archivos, argumentos o parámetros con sus respectivos valores. En los nodos de los argumentos se deben configurar algunas reglas estadísticas, las cuales representan "estados", esto ofrece una descripción de los valores esperados. Para el entrenamiento de la herramienta se hace uso de tráfico artificial, generado a base de diccionarios y peticiones generadas dinámicamente, las

cuales son basadas en el tráfico normal de la aplicación. En este trabajo los autores obtuvieron resultados satisfactorios, debido a que la tasa de alarma de falsos es muy cercana a 0 y la tasa de detección de ataques a 1. Sin embargo, para que tenga resultados satisfactorios en la detección se necesita realizar una configuración correcta del archivo. Para esto, se debe contar con el apoyo de los programadores o administradores del proyecto de desarrollo de la aplicación. Otra de las limitaciones es que se deben considerar las diferentes cabeceras que envían los clientes, ya que hoy en día se cuenta con gran variedad de navegadores, así como reflejar en el archivo los constantes cambios que suelen tener las aplicaciones Web.

Sunitha y Sridevi (2010) proponen un sistema automatizado basado en expresiones regulares para la detección de inyecciones SQL. Otra de las principales características del sistema, es la de marcar los datos que son válidos para la aplicación. El sistema captura los paquetes de la red y se extraen las peticiones HTTP, esto debido a que los paquetes pueden ser de cualquier aplicación de la red. El sistema divide en tokens los datos obtenidos antes de enviarlos a la base de datos, con lo cual se pueden obtener palabras clave, operadores y literales usados en las consultas SQL. El siguiente paso es iterar a través de los tokens obtenidos y comprobar cada uno con las expresiones regulares. Si todos los tokens pasan la prueba, esos datos son considerados seguros. Sin embargo, si las palabras o tokens son palabras en inglés, tales como Select, Insert o Update utilizados como comandos por el SQL y estos no son ataques, pueden generarse falsos positivos. Otra de las limitantes son las expresiones regulares, esto se debe a la complejidad para generarlas.

En el trabajo realizado por Choi y otros (2011) se propone un enfoque para la extracción de características de los códigos maliciosos mediante el uso de n-gramas. Esto para la clasificación de ataques, tales como la inyección SQL, así como los cross site scripting (también conocidos como XSS). Esta clasificación hace uso de un clasificador no lineal, muy

útil en el área de aprendizaje de máquina, conocido como máquina de vectores de soporte. El sistema cuenta con un tokenizador que se encarga de obtener distintas longitudes de n-gramas, el cual está basado en diferentes criterios, y de la realización del conteo de éstos. Los criterios consideran signos de puntuación y palabras clave del lenguaje javascript y de las consultas SQL. Cada n-grama se agrega a un espacio vectorial. Lo anterior corresponde a la etapa de entrenamiento del sistema utilizando código malicioso que fuera obtenido de diferentes fuentes públicas. Al finalizar la etapa de entrenamiento, se cuenta con un vector con el cual se puede trabajar para realizar la comparación de datos. Posteriormente el clasificador SVM se encarga de separar los datos en su respectiva clase. La tasa de precisión haciendo uso de tri-gramas ha sido del 98.04%, pero las pruebas no han sido llevadas a cabo con el uso de una aplicación real y no se comenta en la documentación de la investigación la viabilidad que pueda tener en un entorno de producción. De igual manera, los autores comentan que se debe mejorar el tokenizador para abarcar futuros códigos maliciosos.

En el trabajo realizado por Gallagher y Eliassi-Rad (2009) se presenta un clasificador basado en el modelo de espacio vectorial comúnmente utilizado en recuperación de información. Los autores comentan que este enfoque tiene una serie de características deseables, como robustez para la información contextual, interoperabilidad de los modelos y escalabilidad. El clasificador se encarga de identificar los ataques realizados en las peticiones HTTP. Para el entrenamiento hacen uso del conjunto de datos proporcionado en el evento de minería de datos conocido como ECML/PKDD 2007 Discovery Challenge. La propuesta de los autores es dividir los datos en 8 grupos o documentos, 1 de datos válidos y 7 con los diversos ataques entre los cuales se encuentra la inyección SQL. Cada documento es procesado, obteniendo los tokens y representados como términos de los vectores. El peso de cada término se obtiene haciendo uso de TF-IDF. Para obtener el TF hacen uso de la fórmula:

$$tf(t,d) = \frac{count(t,d)}{\sum_{v \in d} count(v,d)}$$

En donde count(t,d) retorna el número de ocurrencias del término t y $\sum_{v \in d} count(v,d)$ representa la sumatoria de las ocurrencias de los distintos términos en el documento (d).

Para la clasificación, los autores toman cada petición HTTP como una consulta, obtienen los tokens con sus respectivos pesos y hacen uso de la similitud coseno para verificar a que documento es más cercano y así clasificarla. De igual forma, los autores analizan su método con los realizados en el evento, siendo el de ellos el de mayor precisión y eficiencia. Una característica de este método es que corre en un tiempo proporcional al de la petición de entrada. Sin embargo, la herramienta presentada no es probada en una aplicación y en la documentación no se proporciona información acerca de los falsos positivos que se puedan generar y si es posible implementarla en un entorno de producción.

CAPÍTULO IV

Diseño de un Clasificador de Inyecciones SQL

"Saber romper medidas de seguridad no hace que seas hacker, al igual que saber hacer la puerta de un coche no te convierte en un ingeniero automotriz"

Eric Raymond

En el presente capítulo se describe a detalle cada uno de los bloques que compone el clasificador de inyecciones SQL propuesto en el presente trabajo, así como las consideraciones para la selección de las diversas técnicas de las que se hace uso.

4.1 Motivación

Hoy en día, las aplicaciones Web son un gran apoyo en la comunicación y generación de activos para las organizaciones, por lo cual los ataques a éstas han ido avanzando e impactando cada vez más en los activos de las organizaciones. Debido a esto, no solo se debe dejar en las manos de los desarrolladores la seguridad, sino complementar con las herramientas de seguridad de red y las herramientas especializadas para la seguridad en aplicaciones Web.

Se han propuesto un gran número de enfoques, métodos y herramientas para la detección de ataques a aplicaciones Web. La mayoría de estos métodos se enfrentan a dificultades como: baja precisión en la detección, bajo rendimiento en tiempo real, detección de falsos positivos y falsos negativos, mucho tiempo en el análisis del código, y en algunos enfoques (como el del análisis estático) se debe desarrollar una herramienta para cada lenguaje de programación que sirva para el desarrollo de aplicaciones Web. Algunos WAF se basan en expresiones regulares

que deben generar los administradores de la aplicación, pero esto se va haciendo más complicado conforme se van encontrando ataques más elaborados, lo que hace muy difícil mantener estas expresiones. Sin embargo, a pesar de todo lo anterior los WAF siguen siendo un medio proactivo de detección y respuesta a las amenazas cada vez más utilizados cuando se necesita una mayor seguridad en la aplicación o cuando se ha recibido un ataque y no se puede dejar sin servicio a los usuarios mientras se actualiza la aplicación. Esto hace necesario el recurrir a la investigación para poder desarrollar mejores métodos de detección.

4.2 Enfoque y Suposiciones

Nuestro enfoque se basa en las siguientes observaciones; (1) las peticiones HTTP son divididas en campos específicos, lo que las hace semi-estructuradas, sin embargo éstas pueden ser tratadas como un texto sin estructura; (2) los parámetros que son pasados como valores, pueden ser usados por los atacantes para modificar los enunciados SQL de la aplicación inyectando delimitadores, identificadores u operadores SQL; y (3) Para realizar un ataque de inyección SQL el atacante debe inyectar, cuando menos 3 caracteres, operadores o palabras claves del lenguaje de consulta SQL, por lo que se hace uso del modelo de tri-gramas.

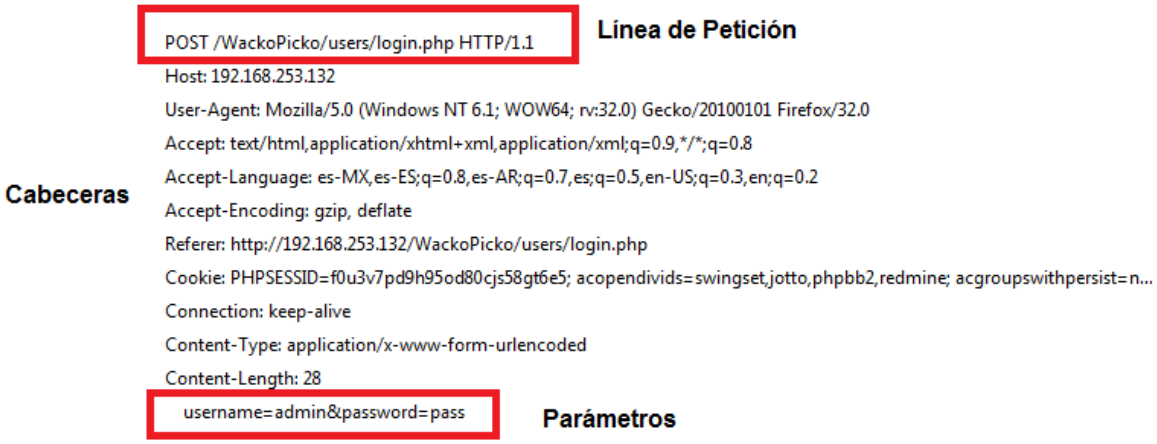

Figura 12. Petición Post con sus respectivas cabeceras y parámetros.

Las fases a seguir especificadas en este trabajo son:

- **Abstraer las peticiones HTTP**. Esto nos permite tratar el problema como uno de clasificación de textos. Para realizar esta abstracción se unirán todas las peticiones de un mismo tipo en un documento, lo cual nos deja con solamente 2 documentos, uno llamado documento Normal en donde se unen todas las peticiones generadas por el uso normal o común de la aplicación y otro documento llamado SQLi, en el cual se agregan todas las peticiones que se realizan con algún ataque de inyección SQL. Se pretende hacer uso de las diversas herramientas en el mercado de ataques de inyección SQL, esto con el fin de contar con una gran cantidad de patrones de ataques que pueden llevarse a cabo.
- **Entrenamiento del clasificador**. Una vez abstraídos los datos, a cada documento se le realiza un proceso para separar los datos en tri-gramas, cada tri-grama es considerado un término o token al cual se le obtiene su peso haciendo uso de TF-IDF.

Clasificación de Inyecciones SQL. Para poder realizar la clasificación se considera cada

petición de entrada como una consulta (simulando la consulta realizada en un buscador en el campo de la recuperación de información). La consulta es procesada, esto quiere decir que se divide en y tri-gramas y se obtiene el peso de cada uno mediante TF-IDF. Para finalizar y obtener a que clase pertenece la consulta o petición se hace uso de la ecuación de similitud coseno.

Uno de los principales objetivos de este trabajo es el reducir los falsos positivos que suelen darse en otras técnicas de clasificación. Éstos pueden ocurrir cuando un valor de parámetro contiene un enunciado o palabra de la sintaxis del lenguaje de consulta SQL, pero este valor no es usado por el desarrollador para formar el enunciado SQL final de la aplicación. En la figura 13, se aprecia que el parámetro "action" tiene como valor la palabra "UPDATE", la cual es parte de la sintaxis de SQL. El problema suele darse al momento de que la palabra sirve dentro de la lógica de la aplicación, aunque no para formar el enunciado SQL el cual es enviado a la base de datos, con lo que no representa ningún peligro, por lo que dicha petición debe ser considerada válida. Para evitar este problema se considera que haciendo uso de los métodos TF-IDF y el modelo de tri-gramas se pueden obtener los pesos de los términos más comunes e importantes que ayuden a evaluar si una petición es peligrosa o normal. Con esto se pueden reducir los falsos positivos que suelen generar en otros métodos.

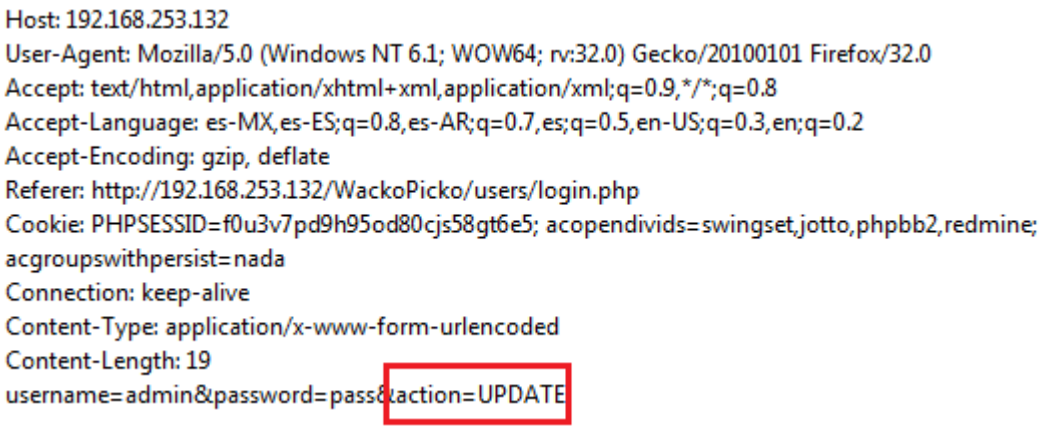

Figura 13. Ejemplo de falso positivo: la palabra UPDATE se encuentra como valor de un parámetro.

4.3 Arquitectura del Clasificador

Para poder llevar a cabo el presente trabajo, se ha desarrollado un WAF en el que se implementa el clasificador propuesto. Esto con el fin de realizar la captura de las peticiones HTTP realizadas por el usuario de la aplicación. Estas peticiones pueden ser clasificadas como inyecciones SQL o peticiones normales.

El sistema WAF es un servidor proxy inverso que se encuentra delante del servidor Web, por lo que recibe las peticiones HTTP realizadas por el usuario de la aplicación antes de que estas lleguen al servidor Web. El WAF no considera el protocolo HTTPS, sin embargo al ser un proxy inverso se puede implementar la captura de este tipo de tráfico.

Para que el WAF pueda clasificar las peticiones como SQLi o de clase Normal, se necesitan dos procesos:

- **Proceso de entrenamiento o aprendizaje**, el WAF debe determinar los criterios de clasificación para cada una de las clases. En esta etapa, se tiene como entrada dos corpus o conjunto de datos, los cuales se procesan con los criterios que se verán más adelante para la extracción y normalización de los datos necesarios para el

aprendizaje.

- **Proceso de clasificación**, el WAF tiene como entrada la petición HTTP realizada por el navegador del cliente, la cual es tomada como una consulta, a la que se le realiza la extracción y normalización de los datos para posteriormente clasificarla.

Para poder implementar el método propuesto en el presente trabajo el WAF cuenta con los siguientes módulos:

- Módulo Interceptor de Peticiones HTTP
- Módulo Generador de Pesos
- Módulo de Clasificación

4.3.1 Módulo Interceptor de Peticiones HTTP

Este módulo intercepta las peticiones HTTP y realiza el filtrado de las cabeceras que aparecen en la tabla 2. Durante la fase de entrenamiento se configura para la captura de peticiones normales y posteriormente la captura de las peticiones con el SQL inyectado. Al llegar una petición, la captura, realiza los determinados filtros y la almacena o agrega al conjunto de datos para el cual ha sido configurado. Al finalizar la fase de entrenamiento, se cuenta con un documento llamado Normal y otro llamado SQLi.

Tabla 3. Cabeceras HTTP filtradas de la petición de entrada.

Cabeceras HTTP
User-Agent
Accept
Accept-Language
Host
Accept-Encoding
Connection
Content-Type
Content-Length

Las cabeceras de la tabla 3 se filtran debido a que suelen tener una frecuente o alta aparición, por lo que no ayudan a discriminar alguno de los documentos. Asimismo, la URL, los parámetros, el cuerpo de la petición y las cabeceras que suelen agregar las herramientas de ataques se almacenan en los documentos para proporcionar una mejor discriminación. Adicionalmente, es importante señalar que las peticiones con las siguientes extensiones no son capturadas, esto se debe a que no puede realizarse una inyección SQL dentro de ellas.

“png | jpeg | jpg | gif | bmp | ico | js | cur | css | mp3 | mp4 | pdf | docx | json | xml | swf | xls | txt”.

En la fase de clasificación, este módulo captura los datos de la petición de entrada a la cual llamamos consulta y realiza los filtros de las cabeceras. Posteriormente envía los datos al módulo para generar pesos.

Conjunto de Datos

Como se menciona en la sección anterior, para poder probar y simular el proceso de detección que realiza el WAF, se necesita un conjunto de datos con los diferentes tipos de inyección SQL y un conjunto de datos libre de ataques. Desafortunadamente, generar estos datos de forma manual es un poco complejo y laborioso. Es por eso, que para generar el conjunto de datos de inyecciones SQL se obtienen peticiones HTTP realizadas por herramientas especializadas en inyección SQL, como son SQLMAP, SQL Inject Me y Havij. Adicionalmente, se ha generado una herramienta para realizar inyecciones que han sido recolectadas de diversos sitios especializados y libros. Se hace uso de éstas herramientas, debido a que generan diversos tipos de inyecciones SQL, que van desde las más básicas a las más avanzadas.

Las herramientas automatizadas se han vuelto populares por la facilidad de uso y que existen muchas que se distribuyen sin ningún costo, es por eso, que son usadas por atacantes novatos, avanzados y por gente que se dedica a realizar pruebas de penetración. Estas inyecciones son realizadas a una aplicación Web vulnerable a propósito. Se ha elegido esta aplicación, ya que se sabe cuáles son las secciones sensibles a un ataque de inyección SQL. Adicionalmente, estas aplicaciones son desarrolladas para simular aplicaciones Web reales. Este tipo de aplicaciones son usadas para evaluar las herramientas de detección (Román, Sabido, & García, 2014). Las peticiones HTTP obtenidas se mezclan para generar un solo documento, el cual conformará nuestro documento de inyección SQL.

Por otra parte, para la generación de tráfico normal en la aplicación, se ha desarrollado una aplicación que simula su uso normal. Para el registro y login de usuarios se cuenta con un diccionario de 3551 nombres diferentes y un diccionario de 3551 passwords, que suelen ser los más usados por los usuarios. Estos datos son tomados por la herramienta, la cual genera

los datos necesarios y realiza las peticiones al servidor Web. De igual forma, se han realizado peticiones desde diferentes navegadores.

4.3.2 Generador de Pesos

Como ya se había mencionado, un peso TF-IDF consiste en dos componentes: TF la frecuencia del término e IDF la frecuencia inversa del documento. En este trabajo le llamaremos término a los trigramas. El componente TF recompensa a los términos que ocurren frecuentemente en un documento, mientras que el componente IDF penaliza a los términos que ocurren en otros documentos.

Esto quiere decir que si la cadena *' or '1'='1* aparece en el documento de inyecciones SQL pero no en el documento normal, esos términos nos ayudarían a detectar que la petición cuenta con una inyección maliciosa de código SQL. Si algún término aparece en ambos documentos el peso que se le asigna es nulo y los otros términos ayudarían a determinar a qué documento pertenece. Esto es debido, a que en la mayoría de las inyecciones SQL se usa la comilla simple.

En las siguientes secciones se verá el proceso para la obtención de los tri-gramas a los cuales se les obtendrán sus pesos para poder realizar la clasificación de inyecciones SQL.

Tokenización

Como se mencionó en el apartado anterior, a los documentos generados por el módulo interceptor de peticiones HTTP se les realiza un proceso denominado tokenización. Este proceso toma su nombre del inglés token cuyo significado es fichas o muestras y consiste en la división del texto en elementos más pequeños o términos (Blázquez, 2013). Por lo general,

los textos suelen dividirse en palabras (cada palabra se considera un token) para posteriormente obtener los pesos de éstas. En el presente trabajo se obtienen palabras y símbolos de los documentos para posteriormente formar el tri-grama o término.

Antes de realizar el proceso de tokenización, se debe realizar un proceso de normalización. Para esto, se realiza la decodificación URL para los datos que se encuentren con éste tipo de codificación. Posteriormente, el texto del documento se convierte a minúsculas para evitar problemas en la distinción de palabras con mayúsculas y minúsculas.

Una vez realizado el proceso de normalización se procede a obtener los tokens del documento, para esto se hace uso de una librería en javascript conocida como natural node (Koch, s.f.). Esta librería realiza la división de las palabras y signos de puntuación del documento para poder trabajar con ellos.

Tri-gramas

Los tri-gramas nos ayudan a realizar una mejor clasificación, esto se debe a las ocurrencias de determinadas secuencias que pueden darse al realizar ataques de inyecciones SQL. Por lo que después de haber obtenido los tokens, se hace uso de un algoritmo para obtener los trigramas de los documentos. Un ejemplo de este proceso es el siguiente:

Supongamos que se tiene la siguiente cadena:

```
id=10' OR '1'='1
```

Se procesa y se obtienen los siguientes tokens:

```
id
=
10'
or
```

```
‘1’
=
’1
```

Y para finalizar se obtienen los siguientes trigramas:

```
id, =, 10’
=, 10’, or
10’, or, ’1’
or, ’1’, =
‘1’, =, ’1
```

Pesos TF-IDF

Para obtener el peso de cada término, se hace un conteo de las ocurrencias de cada término o tri-grama y se hace uso de la fórmula 1.1:

$$wf_{t,d} = \begin{cases} 1 + \log(f(t,d), & si\ f(t,d) > 0 \\ 0, & en\ otro\ caso \end{cases}$$

Para obtener la frecuencia inversa del documento, se divide el número total de documentos en este caso 2 (uno que representa las peticiones permitidas y otro que representa peticiones con inyecciones SQL), por el número de documentos que contienen el término o tri-grama y se toma el logaritmo de ese cociente:

$$idf(t,D) = \log\left(\frac{|D|}{d \in D : t \in d}\right)$$

Teniendo los valores de TF y de IDF, se procede a obtener los valores TF-IDF de cada

término con la fórmula:

$$tf - idf = tf * idf$$

Los términos, junto con los valores TF, IDF y TF-IDF, al igual que el tipo de documento son almacenados en una base de datos para poder recuperarlos en la fase de clasificación.

4.3.3 Módulo de Clasificación

Una vez concluida la fase de entrenamiento, el sistema puede clasificar las inyecciones haciendo uso de los valores generados. Para realizar la clasificación, el módulo interceptor de peticiones recibe la solicitud del cliente, la cual se considera como una consulta en el ámbito de recuperación de información, realiza las validaciones necesarias pero no almacena los datos, sino que los pasa al módulo de generación de pesos. De igual forma que en la fase de entrenamiento, se generan los pesos Tf-IDF de cada tri-grama o término y son pasados al módulo de clasificación.

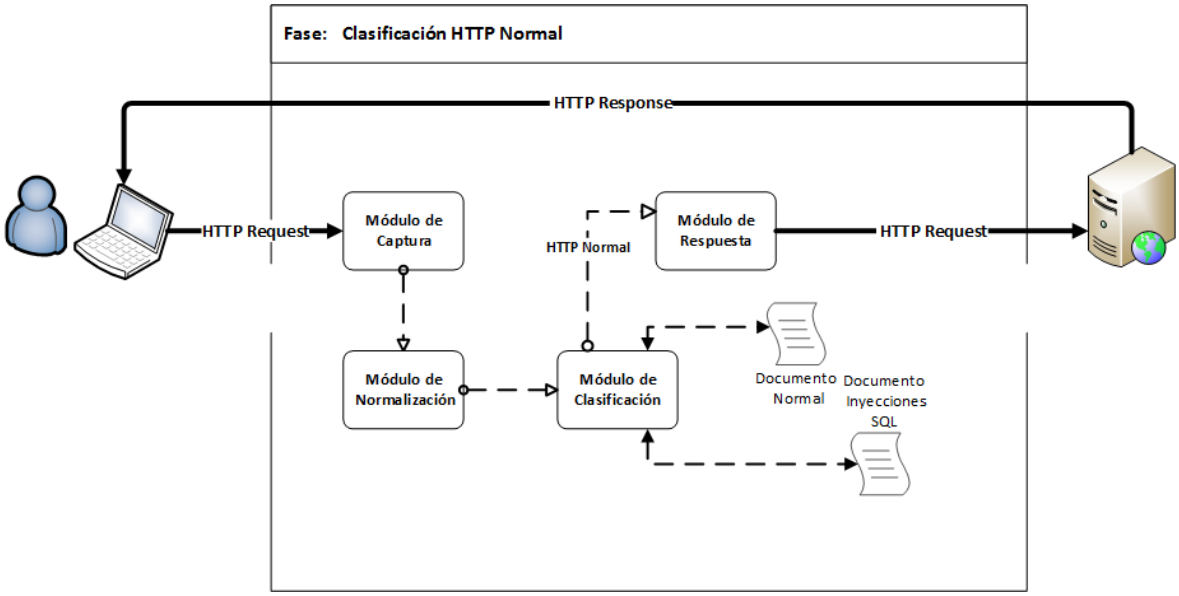

Figura 14. Proceso de clasificación de la petición HTTP.

El módulo de clasificación verifica si existen los tri-gramas, generados de la consulta, en la base de datos y obtiene los pesos de cada uno y el tipo de documento en el que se encuentran. Posteriormente se hace uso de la fórmula del coeficiente de similitud coseno, con los términos obtenidos de la base de datos según el tipo de documento. Se obtienen dos valores, uno para el documento normal y otro para el documento inyectado con inyecciones SQL. El documento con el valor más alto, indica que es el más parecido o relativo a la consulta. Por ejemplo, al realizar el coeficiente de similitud coseno de la consulta con el documento de inyecciones SQL, éste nos ofrece el valor más alto, por lo que se considera que la solicitud enviada por el cliente contiene un ataque.

4.4 Consideraciones

Se debe considerar que el filtrado, análisis y detección de cada petición HTTP requiere un tiempo de procesamiento. Otro punto a considerar es el tiempo que se requiere para el

aprendizaje de los posibles ataques, por lo que el WAF puede afectar en la velocidad de procesamiento o respuesta de la aplicación Web.

Debido a que existen aplicaciones o sistemas Web que requieren procesamientos rápidos o en tiempo real, la implementación de este proxy no es recomendable para este tipo de aplicaciones. Tampoco se debe considerar su implementación en hostings compartidos, debido a que cada aplicación Web maneja distintos datos, lo que haría que el WAF genere muchos falsos positivos y su entrenamiento sea muy complejo. Sin embargo, si se entrena con cada una de las aplicaciones este podría clasificar de forma adecuada los ataques a las diversas aplicaciones del hosting compartido.

La implementación del WAF debe ser considerada en ambientes más controlados, como en una aplicación Web de intranet, en una aplicación universitaria o de una pequeña empresa, donde la rapidez de la aplicación no es muy importante, pero si el evitar que la información sea filtrada o modificada. Al estar en un sistema en producción, se debe designar un día para realizar el entrenamiento para poder registrar los ataques de inyección SQL. Esto con el fin de evitar el filtrado de algún ataque dentro del documento de peticiones normales.

El potencial en la clasificación del WAF depende mucho de los datos proporcionados en el entrenamiento. Mientras más estilos diferentes de ataques SQLi se proporcionen, el sistema podrá ser capaz de detectarlos.

Otro aspecto importante es que en el presente trabajo el WAF sólo considera la codificación URL, pero existen otros tipos de codificaciones que son aceptados como por ejemplo el Hexadecimal, doble codificación URL, etc. Sin embargo, se pueden anexar sin ningún problema estos tipos de codificaciones.

CAPÍTULO V

Implementación y Evaluación del Clasificador

"No harán muy grandes cosas los vacilantes que dudan de la seguridad".

Thomas Stearns Eliot

En este capítulo se hace un análisis sobre los resultados de los experimentos realizados para la clasificación de ataques de inyección SQL, se incluye información relacionada con los ataques generados para el entrenamiento del WAF, así como las diferentes metodologías implementadas para la detección.

5.1 Implementación

Para llevar a cabo la implementación del clasificador se ha decidido implementar el Firewall de Aplicación Web (WAF) en el siguiente entorno:

Servidor WAF

- ✓ Procesador: AMD A10
- ✓ Memoria RAM: 8 GB
- ✓ Disco Duro: 1 TB
- ✓ Sistema Operativo: Windows 7
- ✓ Versión Node.js: 0.10.33
- ✓ Versión MongoDb: 2.6

Servidor Web

- ✓ Máquina virtual VMWare Workstation
- ✓ Memoria RAM: 2 GB

- ✓ Disco Duro: 320 GB
- ✓ Sistema Operativo: Linux Ubuntu
- ✓ Apache
- ✓ PHP
- ✓ MySQL
- ✓ WackoPicko

5.2 Fase de Entrenamiento

El primer paso para el entrenamiento ha sido elegir una aplicación Web vulnerable. Para la elección de la aplicación se ha realizado un proceso de selección basándose en los siguientes puntos:

- La aplicación debe tener claramente definidas las vulnerabilidades, en especial la de inyección SQL.
- Debe ser fácil de personalizar y agregar nuevas vulnerabilidades.
- Contar con tecnologías y funcionalidades de aplicaciones Web actuales.
- Contar con buena documentación.

Considerando los puntos mencionados, se ha decidido hacer uso de la aplicación conocida como WackoPicko, la cual ha sido desarrollada en el trabajo Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners. Esta aplicación está desarrollada en php y hace uso de la base de datos MySQL. Su funcionalidad se basa en un portal de venta de fotos, la cual la hace ser una aplicación basada en un entorno real. Cuenta con varios tipos de vulnerabilidades, entre ellas las inyecciones SQL. Ésta aplicación cuenta con una buena documentación de las vulnerabilidades que se encuentran en ella.

Ya elegida la aplicación, se necesita capturar una gran cantidad de peticiones HTTP con inyecciones SQL y normales para generar los 2 documentos. Realizar las inyecciones SQL de forma manual puede llegar a ser un proceso complejo y que requiere de bastante tiempo para llevarse a cabo, por lo que surgió la necesidad de encontrar aplicaciones que automaticen este proceso. Para ello, se recurre a 3 aplicaciones especializadas en Inyecciones SQL, las cuales son ampliamente conocidas y usadas por los profesionales de la seguridad, así como por atacantes. Una ventaja que nos ofrece el hacer uso de estas herramientas, es que si el clasificador ya se encuentra entrenado con las inyecciones generadas por éstas, puede haber una mayor probabilidad de detectar el ataque llevado a cabo por un atacante con alguna de éstas.

Las aplicaciones seleccionadas para la fase de entrenamiento son las siguientes:

- **Havij**. Es una herramienta de inyección SQL automatizada que ayuda a los profesionales de la seguridad a encontrar y explotar vulnerabilidades de inyección SQL en una página Web. El poder distintivo de Havij, que lo diferencia de otras herramientas similares, reside en sus métodos únicos de inyección. La tasa de éxito del ataque a objetivos vulnerables utilizando. Havij está por encima del 95%.
- **SqlMap**. Es una herramienta de pruebas de penetración de código abierto que automatiza el proceso de detectar y explotar los errores de inyección SQL y hacerse cargo de los servidores de bases de datos. Soporte completo para seis técnicas de inyección SQL: boolean-based blind, ime-based blind, basado en error, consulta de unión, consultas apiladas y out of band.
- **Herramienta a la medida**. Durante el presente trabajo se ha desarrollado una herramienta para realizar inyecciones SQL. Para esto, se ha recopilado en diversos

portales de seguridad, libros y blogs ataques que suelen realizarse a las aplicaciones Web, con lo cual se tienen 111 ataques distintos.

- **SQL Inject Me**. Esta herramienta es un Plugin del navegador Firefox. Funciona mediante la presentación de los formularios HTML y sustituyendo el valor formulario con cadenas que son representativas de un ataque de inyección SQL.

De igual forma, para la simulación de peticiones normales se ha desarrollado una herramienta que hace uso de diccionarios. Los diccionarios contienen datos que suelen usar los usuarios que se registran en diversas plataformas Web, como son nombres de usuario, contraseñas, nombres y apellidos. Estos diccionarios son generados por algunos portales de internet relacionados a la seguridad.

Durante la fase de entrenamiento se configuraron las herramientas para hacer uso del proxy desarrollado e interceptar las peticiones de éstas. Los ataques a la aplicación fueron realizados a los distintos formularios, como los de login de usuario (ver figura 16), login de administrador y el de la sección de mensajes.

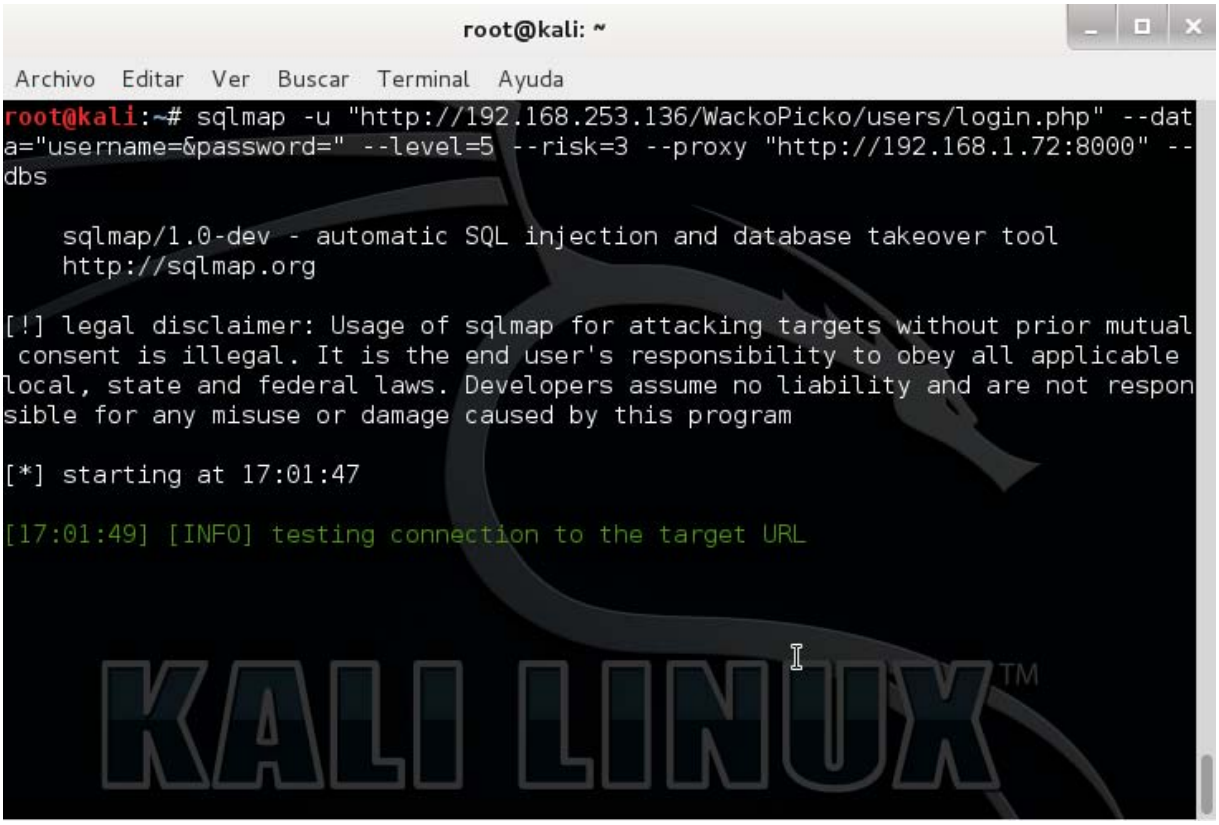

Figura 15. SqlMap

En la figura 15, se puede apreciar la configuración de SqlMap para llevar a cabo el ataque al formulario de login. Esta herramienta genera distintos tipos de ataque de inyección SQL, así como valores aleatorios que son necesarios, por ejemplo: nombre de tablas, nombre de usuarios, etc. De igual forma, hace ataques con palabras codificadas en hexadecimal y URL.

En el siguiente ejemplo de petición capturada por el proxy, se tiene un ataque de inyección SQL realizada por la herramienta SqlMap:

```
POST http://192.168.253.136:80/WackoPicko/users/login.php accept-charset:
ISO-8859-15,utf-8;q=0.7,*;q=0.7 cookie:
PHPSESSID=qebus9h83vdeefoqp495869pb3 pragma: no-cache cache-control: no-
cache,no-store username=' AND (SELECT 7496 FROM(SELECT
COUNT(*),CONCAT(0x7179716d71,(SELECT MID((IFNULL(CAST(last_login_on AS
```

```
CHAR),0x20)),1,50)    FROM    wackopicko.users    ORDER    BY    id    LIMIT
43,1),0x71756e7071,FLOOR(RAND(0)*2))x                                    FROM
INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a) AND 'xybP'='xybP&password=
```

En la siguiente imagen se puede apreciar uno de los formularios que ha sido atacado por las herramientas. Así mismo, se ha configurado la herramienta para generar peticiones normales basadas en este formulario.

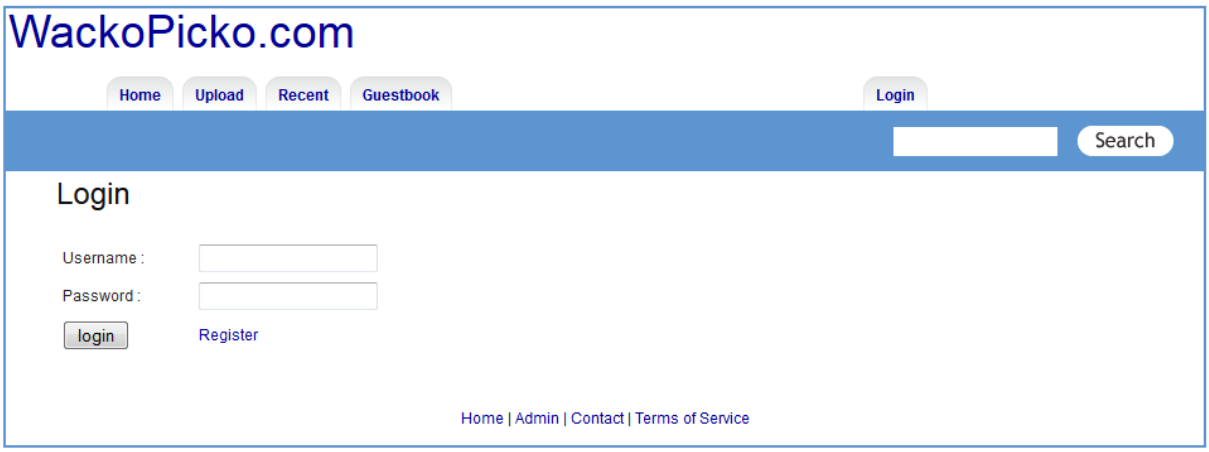

Figura 16. Formulario de inicio de sesión de usuario en WackoPicko.

Para poder realizar la simulación del uso de la aplicación WackoPicko, se ha configurado la herramienta desarrollada para registrar a 3551 usuarios distintos haciendo uso del formulario de registro de WackoPicko. Posteriormente, se configuraron los distintos formularios para realizar las peticiones con los datos de los usuarios registrados. Para finalizar, se ha navegado la aplicación haciendo uso del navegador Firefox y llenando algunos formularios con palabras clave del lenguaje SQL, esto con el fin de capturar cabeceras que sean generadas por este navegador y poder evaluar los falsos positivos. Un ejemplo de petición para el registro de usuario puede observarse a continuación:

```
POST
/WackoPicko/users/register.php
username=paule&password=77867786&againpass=77867786&firstname=darnall&lastn
ame=downey
```

Debido a la cantidad de peticiones que generan las herramientas (sobretodo SqlMap), solo han sido configuradas para realizar un escaneo en los formularios que no requieren haber iniciado sesión. A excepción de SQL Inject Me y la herramienta desarrollada, estas han sido configuradas para realizar análisis o ataques a cada formulario.

Al finalizar los ataques y la simulación del uso de la aplicación Web, se obtienen los resultados de la tabla 4:

Tabla 4. Cantidad de peticiones capturadas.

Tipo	Peticiones Capturadas
Normal	95,018
SQLi	12,715

En la siguiente tabla (ver tabla 4), se aprecian los Tri-gramas totales que han sido generados por cada documento. Una vez que ha sido realizado el entrenamiento y que se han obtenido los pesos, puede observarse, en la tabla 5, la cantidad de Tri-gramas diferentes que hay en cada documento. Se observa, que el documento normal cuenta con una menor cantidad de Tri-gramas unidos comparado con el documento de inyecciones SQL. De igual forma, se puede apreciar que el peso del documento Normal es menor que el peso del documento SQLi, a pesar de haber guardado más peticiones (ver tabla 5). Esto último, se debe a que los ataques de inyección SQL generan cadenas de texto más largas que las que comúnmente se usan en la

aplicación Web.

Tabla 5. Tri-gramas obtenidos en la fase de entrenamiento.

Documento	Cantidad de Tri-gramas en el documento	Cantidad de Tri-gramas Únicos	Peso del documento
SQLi	**1,464,924**	**79,769**	**4.78 MB**
Normal	**1,331, 810**	**22,025**	**4.91 MB**

El proceso de entrenamiento genera 2 archivos de texto (Sqli.txt y Normal.txt), los cuales contienen las peticiones capturadas en la fase de entrenamiento. A continuación, se puede ver una parte del archivo Sqli.txt generado:

Figura 17. Archivo Sqli.txt

En la figura 17, se observa el contenido del archivo Sqli.txt, el cual hace posible la generación de Tri-gramas y pesos TF-IDF. Para finalizar la etapa de entrenamiento, se leen los

documentos generados y se obtienen los datos necesarios. Posteriormente, los datos se almacenan en la base de datos tal como se muestra en la figura 18.

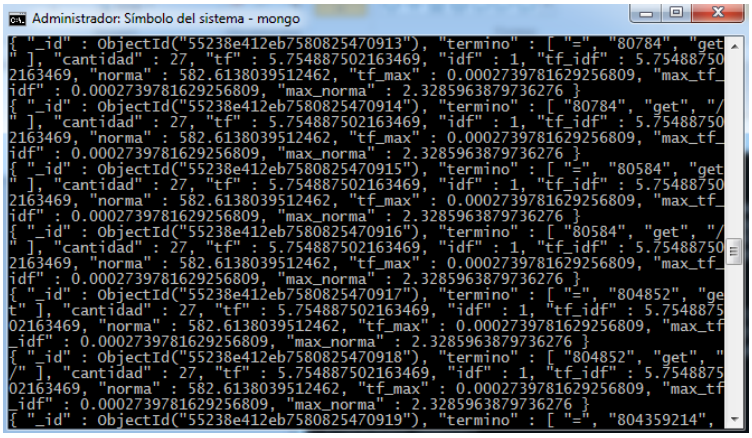

Figura 18. Generación de Pesos.

5.3 Proceso de Clasificación

Una vez finalizada la etapa de entrenamiento y generación de pesos, se realiza la etapa de pruebas. En esta etapa el proxy captura las peticiones y las clasifica como Inyección SQL o Normal. El tráfico que se genera en la etapa de pruebas contiene peticiones normales, así como peticiones con inyecciones SQL.

Para las pruebas se hace uso de la aplicación Web desde los navegadores: Chrome, Firefox (usado en la etapa de entrenamiento), Opera e Internet Explorer. De igual forma, en los formularios se hace uso de frases con palabras tales como: UPDATE, SELECT, DELETE, UNION e INSERT; esto con el fin de intentar engañar al clasificador, ya que esas palabras son parte del lenguaje de consulta SQL. Para los experimentos con peticiones que contienen inyecciones SQLi, se hace uso de las herramientas: Owasp ZAP y VEGA. Estas herramientas generan sus propios ataques de inyección, así como peticiones que pueden ser normales, esto sucede en la etapa conocida como crawling. El crawling es necesario para obtener

información de la aplicación, tal como puede ser formularios y cookies generadas, así como también verifica el comportamiento de dicha aplicación que puedan darse. Como ejemplo de comportamiento se tienen los mensajes de error o cambios en los textos de los formularios.

Estas pruebas se hacen con el fin de verificar si el clasificador es capaz de detectar ataques realizados por aplicaciones de uso profesional y ataques con los que no haya sido entrenado. Del mismo modo, se hicieron ataques con las herramientas Havij, SQL Inject Me y SqlMap, con el fin de verificar que el clasificador sea capaz de detectar las inyecciones con las que ha sido entrenado, sin arrojar falsos positivos o negativos.

En la figura 19, se puede observar que la herramienta Vega ha detectado una vulnerabilidad que hace posible la inyección SQL en el formulario de login de la aplicación WackoPicko.

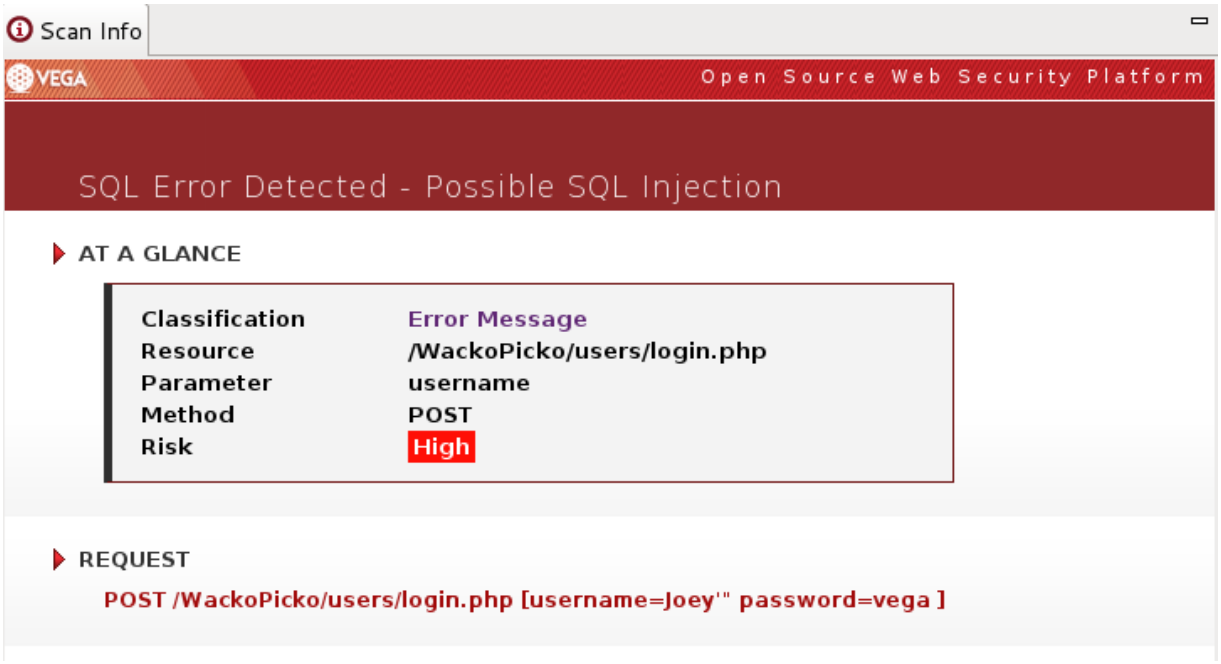

Figura 19. Inyección SQL detectada en WackoPicko por la herramienta Vega.

5.3.1 Falsos Positivos

El principal objetivo de este trabajo es el de poder clasificar las inyecciones SQL evitando en

lo mínimo los falsos positivos. Para esto, ha sido necesario realizar las pruebas separando las herramientas y los datos, en conjunto de datos para pruebas (herramientas VEGA y OWASP Zap) y conjunto de datos de entrenamiento (herramientas con las que se hicieron ataques en la fase de entrenamiento). A continuación se dan a conocer los resultados que han sido obtenidos.

Conjunto de Datos para Pruebas

En la tabla 6, se puede observar el resumen de los datos que se obtiene de las peticiones realizadas por las dos herramientas especializadas en la detección de vulnerabilidades Web. En la primera columna se encuentra el nombre de la herramienta, seguido del total de peticiones realizadas (tanto normales como ataques de inyección SQL) y las peticiones con algún tipo de ataque de inyección SQL. Posteriormente, se tiene la columna que muestra las peticiones que han sido clasificadas como inyecciones SQL por el clasificador, las peticiones SQL detectadas correctamente por el clasificador y la columna de falsos positivos, en la que se aprecia la cantidad de peticiones clasificadas como un ataque, sin embargo, puede ser considerada como una petición normal en la aplicación Web. En la última columna se aprecia el porcentaje de inyecciones SQL detectadas correctamente.

Tabla 6. Resumen de resultados obtenidos para falsos positivos en el conjunto de datos para pruebas.

Página	Total Peticiones	Peticiones SQLi	Peticiones SQLi Clasificadas	Peticiones SQLi Detectadas	Falsos Positivos	Porcentaje Inyecciones Detectadas
Vega	668	369	153	137	16	37.12
Owasp Zap	608	395	578	384	194	97.2

De acuerdo a los resultados de la tabla 6, en el caso de la herramienta Vega, se alcanza una clasificación de inyecciones muy baja (37.12% de precisión en la detección de inyecciones SQL), esto se debe a que la herramienta es un escáner, por lo que la mayoría de las inyecciones que realiza son enfocadas en obtener algún error en la aplicación para verificar si ésta es vulnerable y no para obtener información de la base de datos o manipularla de forma que pueda haber una pérdida de información, por lo que las inyecciones suelen ser cadenas muy cortas o de un solo carácter, lo cual dificulta la detección.

Un ejemplo de lo mencionado puede ser:

```
No es inyección. Normal: 0.03615976134504049 SQLi: 0.03393669533663379 POST
/WackoPicko/users/login.php proxy-connection: Keep-Alive cookie:
PHPSESSID=3i3k1sc9v8bqkfst5oh27s6dq5 cookie2: $Version=1
username=Joey'&password=vega
```

En el ejemplo anterior, se aprecia que se clasifica como una petición normal, pero se puede observar en el parámetro *username* la inyección de una comilla simple. Esta comilla, tal como se menciona en el capítulo 2, suele ser usada para verificar si la aplicación es vulnerable a los ataques de inyección SQL. Así mismo, Vega se especializa en la detección de inyecciones SQL ciegas, por lo que muchas de estas no han sido detectadas debido a que las aplicaciones con las que fue entrenado el clasificador no se especializan en este tipo de ataques. Un ejemplo de este tipo de inyección, podría ser el de verificar si el comando SLEEP puede ser ejecutado:

```
No es inyección. Normal: 0.034575345013037044 SQLi: 0.032449687338087854
POST /WackoPicko/users/login.php proxy-connection: Keep-Alive cookie:
PHPSESSID=3i3k1sc9v8bqkfst5oh27s6dq5 cookie2: $Version=1 username=1 AND
```

```
SLEEP(30) -- &password=vega
```

En el caso de los 16 falsos positivos, se debe a que la herramienta Vega genera valores aleatorios para las cabeceras o directorios con el fin de analizar el comportamiento de la aplicación, en otras ocasiones no envía los parámetros necesarios para ser clasificados como una petición normal o los envía vacíos. Estos valores, se deben considerarse peticiones normales debido a que no pueden ser considerados ataques de inyección SQL. Sin embargo, al no encontrarse dentro del documento normal, el clasificador no puede dar una respuesta a favor de éste documento. A continuación se presentan algunos casos de este tipo:

Ejemplo 1. La herramienta genera un directorio de forma aleatoria:

```
Es inyección. Normal:  0.006466424800482646 SQLi: 0.006466424800482646 GET
/WackoPicko/~nosuchpage123     proxy-connection:     Keep-Alive     cookie:
PHPSESSID=3i3k1sc9v8bqkfst5oh27s6dq5 cookie2: $Version=1
```

Se aprecia que el clasificador les asigna el mismo valor de similitud, a Normal y SQLi, por lo que como regla del clasificador ésta es considerada una inyección. Esto se debe a que no se tienen los datos suficientes en el documento normal para poder asignarle un mayor peso.

Ejemplo 2. Valor de parámetro desconocido:

```
Es inyección. Normal:  0.0061654941334300965 SQLi: 0.0061654941334300965
GET   /WackoPicko/?page=asdf6089   proxy-connection:   Keep-Alive   cookie:
PHPSESSID=9pn3p0j8uv7v9n91q94jv9n7r7 cookie2: $Version=1
```

De igual forma que el ejemplo anterior, se observa que la herramienta Vega genera un valor para el parámetro *page*, sin embargo este valor de parámetro no se encuentra en el

documento Normal, por lo que no se le asigna un peso que pueda ayudar a clasificarlo como Normal. En esta forma, el documento Normal funciona como una lista blanca en la que solo se permiten peticiones válidas para la aplicación.

El caso de los falsos positivos de OWASP Zap, es muy parecido a lo que sucede en Vega. Sin embargo, se puede apreciar en la tabla 6, que con la herramienta Owasp Zap se tiene una efectividad del 97.2%, siendo un escáner de vulnerabilidades igual que Vega. Analizando los datos generados por Owasp Zap, se observa que no solo genera inyecciones para verificar si la aplicación es vulnerable, sino que genera inyecciones más sofisticadas para extraer datos de la base de datos

A continuación se tiene un ejemplo de petición clasificada como inyección:

```
Es inyección. Normal:  0.06611714099354496 SQLi: 0.10918713546863336 POST
http://192.168.253.136/WackoPicko/admin/index.php?page=login  pragma:  no-
cache     cache-control:     no-cache     proxy-connection:     Keep-Alive
adminname=ZAP&password=ZAP
```

En el ejemplo, se puede apreciar que no existe una inyección SQL, sin embargo, la herramienta ha generado un nombre de usuario y contraseña aleatorio (ZAP). Estos datos aleatorios no se encuentran en el documento Normal, debido a que la aplicación WackoPicko cuenta con un usuario administrador con diferentes credenciales de inicio de sesión, por lo que la petición ha sido clasificada como inyección.

Conjunto de Datos de Entrenamiento

En la tabla 7 se cuenta con la misma información presentada en la tabla 6, pero para el

conjunto de datos de entrenamiento. En este conjunto, podemos encontrar las peticiones capturadas por el proxy en la fase de entrenamiento. En la tabla 7, se observa que al realizar los ataques con las diferentes herramientas, las cuales han sido utilizadas en la fase de entrenamiento, se ha obtenido un porcentaje de 100% en la tasa de detección de las inyecciones SQL. Se debe mencionar, que al momento de realizar el entrenamiento haciendo uso de la herramienta SqlMap se configuró para generar ataques de nivel 1, pero para las pruebas se ha configurado para realizar ataques de nivel 5, con lo cual la herramienta realiza una mayor cantidad de ataques más elaborados. A pesar de la diferencia en la configuración, el clasificador ha sido capaz de distinguir los ataques.

Tabla 7. Resumen de resultados obtenidos para falsos positivos en el conjunto de datos de entrenamiento.

Página	Total Peticiones	Peticiones SQLi	Peticiones SQLi Clasificadas	Peticiones SQLi Detectadas	Falsos Positivos	Porcentaje Inyecciones Detectadas
Inject Me	272	272	272	272	0	100
SqlMap	18076	18076	18076	18076	0	100
Havij	1116	1116	1116	1116	0	100
Herramienta desarrollada	336	336	336	336	0	100

En la tabla 7, se puede observar que no se generan falsos positivos. Al ser herramientas de ataque, no realizan un *crawling* como los escáneres de vulnerabilidades Web, por lo que no envían peticiones, que puedan ser consideradas normales, para verificar el comportamiento de la aplicación.

5.3.2 Falsos Negativos

En esta apartado se hace un análisis de los falsos negativos que se pudieran dar en el proceso de clasificación. Estos ocurren cuando un ataque no es reconocido y la herramienta permite el paso de la petición.

Conjunto de Datos para Pruebas

El resumen de datos generados por el clasificador al momento de clasificar las peticiones Normales, que han sido realizadas por las dos herramientas especializadas en la detección de vulnerabilidades, se pueden observar en la tabla 8. En la primera columna, se encuentra el nombre de la herramienta, seguido del total de peticiones realizadas (tanto normales como ataques de inyección SQL) y las peticiones consideradas normales. Posteriormente, se tiene la columna que muestra las peticiones que han sido clasificadas como Normales por el clasificador, las peticiones Normales detectadas o clasificadas correctamente por el clasificador y la columna de falsos negativos, en la que se aprecia la cantidad de peticiones clasificadas como una petición normal o legítima, sin embargo debería ser clasificada como un ataque a la aplicación Web. En la última columna se aprecia el porcentaje de peticiones Normales o legítimas clasificadas correctamente.

Tabla 8. Resumen de resultados obtenidos para falsos negativos en el conjunto de datos para pruebas.

Página	Total Peticiones	Peticiones Normales Realizadas	Peticiones Normales Detectadas	Peticiones Normales Clasificadas	Falsos Negativos	Porcentaje Normales Detectado Correctamente
Vega	668	299	283	515	232	**92,8**
OWASP Zap	608	213	19	30	11	**8.9**

En la tabla 8, se puede apreciar un elevado índice de falsos negativos lo cual hace que se tenga un bajo porcentaje en la detección de inyecciones SQL (como puede observarse en la tabla 6), pero como se menciona en la sección anterior, muchos de estos ataques son para verificar si la aplicación Web es vulnerable a inyecciones SQL y estos ataques suelen ser la inyección de algún simple carácter como la comilla simple o doble y la barra invertida. De igual forma, puede darse el caso de que la herramienta realice algún tipo de inyección que genera un Tri-grama que no se encuentra en la base de datos y al ser un ataque de longitud corta no se generan varios Tri-gramas que puedan dar un mayor peso al documento de inyecciones SQL.

En el caso del porcentaje detectado correctamente como una petición legítima o normal con la herramienta Vega, se tiene un 92.8%, esto significa que el clasificador ha sido capaz de reconocer peticiones tales como visitas a las diferentes secciones de la aplicación Web, formularios que han sido llenados con datos correctos (como el formulario de registro), etc. El caso de OWASP se debe a la gran cantidad de falsos positivos, véase tabla 6, que han sido generados.

Tabla 9. Resumen de resultados de peticiones realizadas con diferentes Navegadores Web.

Navegador	Total Peticiones	Clasificación Normal	Falsos Positivo	Porcentaje Clasificado Normal
Chrome	236	184	52	78
Opera	211	158	53	74,9
Firefox	230	227	3	98,7
IE	246	232	14	94,3

En la tabla 9, se pueden apreciar los resultados obtenidos al navegar la aplicación Web desde distintos clientes (navegadores Web), existentes en el mercado. Los resultados de los navegadores Chrome y Opera, se deben a las diferentes cabeceras que envían en cada petición en contraste al navegador Firefox, con el que ha sido entrenado el clasificador. Los 3 falsos positivos detectados con el uso de Firefox se deben a que no se tiene la cantidad suficiente de Tri-gramas de algunos enlaces o secciones de la aplicación Web. Esto último, se relaciona con la mayor cantidad de peticiones realizadas en la fase de entrenamiento al momento de capturar las inyecciones SQL para esos enlaces. Estos resultados pueden mejorarse con la herramienta de navegación desarrollada, para esto se hace necesario el tener que recopilar las cabeceras usadas por los navegadores y realizar las peticiones con estas cabeceras. De igual forma, se debe intentar tener la misma cantidad de peticiones capturadas por cada sección de la aplicación Web. Otro punto importante que debe mencionarse, es que al momento de realizar la navegación en los formularios se ha capturado información que contienen palabras como INSERT, SELECT, UPDATE, DELETE y estas han sido clasificadas como peticiones normales, lo cual valida nuestra propuesta de hacer uso de Tri-gramas.

Conjunto de Datos de Entrenamiento

En el caso del conjunto de datos de Entrenamiento no se cuenta con falsos negativos debido a que se ha obtenido un 100% en la clasificación de las inyecciones SQL.

A continuación se tienen las conclusiones obtenidas al realizar el presente trabajo, así como las aportaciones realizadas y trabajo a futuro.

CAPÍTULO VI

Conclusiones

"Ser lo que soy, no es nada sin la seguridad"

W. Shakespeare

En este capítulo se menciona la importancia de los resultados adquiridos durante el desarrollo del trabajo, así como las contribuciones al campo de la seguridad en la detección de inyecciones SQL.

6.1 Replanteamiento de la Hipótesis

El principal objetivo del presente trabajo puede ser expresado con la confirmación de la siguiente hipótesis:

"El uso del clasificador de similitud de texto basado en el modelo vectorial, así como en el método de Tri-Gramas, puede ayudar a clasificar los ataques de inyecciones SQL a las aplicaciones Web con una mayor precisión".

Para que un clasificador alcance una alta precisión se debe evitar que un ataque sea clasificado como una petición legítima (falso positivo), así como que una petición legítima sea clasificada como un ataque (falso negativo).

Con el objetivo de comprobar la hipótesis, se ha desarrollado un WAF basado en el método de clasificación propuesto. El WAF se ha instalado delante de una aplicación Web conocida como WackoPicko, la cual es un portal para la venta de fotos.

6.2 Resultados

El análisis realizado en el capítulo 5 plantea los escenarios más comunes que pueden generar falsos positivos o negativos. En estos resultados, se puede concluir que el clasificador propuesto ha alcanzado una alta precisión en la clasificación de ataques sofisticados o que sirven para hacer daño a la aplicación Web. Sin embargo, en las inyecciones enfocadas a obtener algún error en la aplicación, tal como se explica en el capítulo 2, se han obtenido resultados bajos. Esto es debido, a que son cadenas muy cortas o de un solo carácter, por lo que el método de Tri-gramas no resulta adecuado para éstos.

Otro de los objetivos, ha sido el de verificar los falsos negativos que pudieran darse con el uso de palabras clave que son parte del lenguaje SQL. Para esto, se han capturado diferentes frases en los formularios con palabras como: INSERT, DELETE, UPDATE y SELECT. En este caso, podemos concluir que se logran evitar los falsos negativos con el uso de los Tri-gramas.

Los falsos negativos que se han obtenido con el uso de diferentes navegadores de internet, se debe a las diferentes cabeceras enviadas de cada navegador. De igual forma, algunas de estas cabeceras suelen ser usadas por las herramientas de inyección o no se encuentran en el documento normal, por lo que no ofrecen una mayor relatividad con este documento. Sin embargo, esto puede solucionarse agregando las distintas cabeceras a la herramienta desarrollada para la navegación automática o capturando los datos desde diferentes navegadores.

6.3 Conclusiones

Durante la realización del presente trabajo, se ha podido apreciar que una de las principales

dificultades ha sido la recolección de datos, sobre todo para poder clasificar de forma correcta las peticiones normales realizadas por distintos navegadores. Sin embargo, esto puede solucionarse agregando las distintas cabeceras, que suelen enviar dichos navegadores, a la herramienta desarrollada, en el presente trabajo, que simula la navegación automática. Para los datos que contienen inyecciones SQL se hace uso de las herramientas más comunes en el ámbito de seguridad Web, tales como Havij, InjectMe y SqlMap.

Debido a que el clasificador se ejecuta en tiempo proporcional al tamaño de la petición de entrada, se puede llegar a alentar el servicio ofrecido por la aplicación. Sin embargo, mediante técnicas de procesamiento en paralelo o sistemas distribuidos puede dividirse el trabajo del clasificador para mejorar el tiempo de ejecución. Existen algunos servicios en la nube que se basan en estas técnicas para distribuir el trabajo y actuar como un proxy permitiendo evitar ataques de denegación de servicio, un buen ejemplo puede ser CloudFlare.

La implementación del WAF debe ser considerada en ambientes más controlados, como en una aplicación Web de intranet, en una aplicación universitaria o de una pequeña empresa, donde la rapidez de la aplicación no es muy importante, pero si el evitar que la información sea filtrada o modificada.

Al estar en un sistema en producción, se debe designar un tiempo para realizar el entrenamiento para poder registrar los ataques de inyección SQL, esto con el fin de evitar el filtrado de algún ataque dentro del documento de peticiones normales.

6.4 Comparación con Estrategias Existentes

Durante el análisis realizado en el capítulo 3, se pudo observar que existen diversos métodos basados en expresiones regulares, minería de datos, agentes inteligentes, redes

neuronales, etc. Sin embargo, estos métodos suelen generar muchos falsos positivos, cuando los parámetros cuentan con alguna palabra clave contenida en el lenguaje SQL, tal como: INSERT, UPDATE, UNION, SELECT y DELETE. En el clasificador propuesto, esto no ocurre debido a que los Tri-gramas se basan en ocurrencias de determinadas secuencias que pueden darse en los ataques de inyección SQL.

Otros enfoques, como los basados en expresiones regulares, son muy complejos de configurar y difíciles de mantener. Con el enfoque propuesto, se puede apreciar que la configuración es simple y podría ser extendido para otros tipos de ataques.

De igual forma, se han desarrollado métodos para realizar un análisis estático del código fuente pero esto hace necesario desarrollar una herramienta para cada lenguaje de programación. Con el enfoque basado en las peticiones HTTP no es necesario el analizar el código de la aplicación.

6.5 Contribuciones

Las contribuciones del presente trabajo son las siguientes:

- El clasificador basado en el modelo vectorial y de Tri-gramas.
- Desarrollo de una herramienta para inyección SQL.
- Análisis de las características que deben tener las aplicaciones Web vulnerables, que tienen como objetivo, comprobar las capacidades de las herramientas de análisis de vulnerabilidades.
- Propuesta para la generación de datos para el clasificador.
- Análisis de la peligrosidad de los ataques de inyección SQL.

6.6 Trabajo Futuro

Para investigaciones futuras se pretende implementar el clasificador para otros tipos de ataques a aplicaciones Web, tales como cross site scripting, directory traversal, inyección no-SQL. Así mismo, se pueden analizar otros métodos de clasificación en el área de recuperación de información para la detección de estos tipos de vulnerabilidades.

Otro campo interesante, puede ser el de las herramientas para la detección de vulnerabilidades, las cuales pueden ayudar a los desarrolladores a encontrar las vulnerabilidades antes de que el sistema se encuentre en producción. Una de las principales dificultades de las herramientas, que se ha apreciado durante el desarrollo del presente trabajo, es la navegación para obtener la información de la aplicación Web, como son los formularios y cookies.

Finalmente, como siguiente paso de este trabajo, es el realizar una evaluación del tiempo de procesamiento de los datos y proponer una forma de mejorar el desempeño.

Bibliografía

Beechey, J. (2009). *Web Application Firewalls: Defense in Depth for Your Web Infrastructure.* SANS. Retrieved from http://www.sans.edu/resources/student_projects/200904_01.doc

Blázquez, M. (2013). *Técnicas avanzadas de recuperación de información, Procesos, técnicas y métodos.* Madrid: mblazquez.es.

Choi, J., Kim, H., Choi, C., & Kim, P. (2011). *Efficient Malicious Code Detection Using N-Gram Analysis and SVM.* Tirana: International Conference on Network-Based Information Systems.

Clark, J. (2009). *SQL Injection Attacks and Defense.* Burlington: Syngress Publishing, Inc.

Croos, M., Kapinos, S., Meer, H., Muttik, I., Palmer, S., & Petkov, P. (2007). *Web Application Vulnerabilities Detect, Exploit, Prevent.* Burlington: Syngress Publishing, Inc.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (Junio de 1999). *Hypertext Transfer Protocol*. Recuperado el 20 de Mayo de 2014, de RFC 2616: http://tools.ietf.org/html/rfc2616

Gallagher, B., & Eliassi-Rad, T. (2009). *Classification of HTTP Attacks: A Study on the ECML/PKDD 2007 Discovery Challenge.* Lawrence Livermore National Laboratory. Livermore: Lawrence Livermore National Laboratory Technical Report LLNL-TR-414570 (Also presented at Center for Advanced Signal and Image Sciences (CASIS) Workshop.

Graves, K. (2007). *CEH Official Certified Ethical Hacker Review Guide.* Indianapolis: Wiley Publishing, Inc.

Greengrass, E. (2000). *Information Retrieval: A Survey.* Baltimore: University of Maryland, Baltimore County.

Gupta, N., & Saikia, A. (2007). *WEB APPLICAT ION FIREWALL.* Kanpur, India: Department of Computer Science and Engineering, Indian Institute Of Technology.

Imperva. (2014). *WEB APPLICATION ATTACK REPORT #5.* IMPERVA. Redwood Shores, CA: Imperva.

Koch, K. (s.f.). *NaturalNode*. Recuperado el 20 de Noviembre de 2013, de Github: https://github.com/NaturalNode/natural

Liao, Y., & Vemuri, R. (2002). *Using Text Categorization Techniques for Intrusion Detection.* University of California, Department of Computer Science,. Berkeley: ACM, 11th USENIX Security Symposium.

Manning, C., Raghavan , P., & Schütze, H. (2008). *Introduction to Information Retrieval.* 2008 Cambridge University Press. Obtenido de http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html

Moosa, A. (2010). Artificial Neural Network based Web Application Firewall for SQL Injection. *World Academy of Science, Engineering and Technology*, 12-21.

NIST. (2008). *Volume I: Guide for Mapping Types of Information and Information Systems to Security Categories.* National Institute of Standards and Technology, Computer Security Division. Gaithersburg: NIST Special Publication.

OWASP. (2008). *Guía de Pruebas OWASP.* OWASP Foundation.

OWASP. (4 de Diciembre de 2012). *Inyección SQL*. Recuperado el 2013 de Septiembre de 22, de Categoría de Ataques: https://www.owasp.org/index.php/Inyecci%C3%B3n_SQL

OWASP. (2013). *OWASP Top 10 - 2013. Los diez riesgos más críticos en Aplicaciones Web.* USA: OWASP. Obtenido de https://www.owasp.org/images/5/5f/OWASP_Top_10_-

_2013_Final_-_Espa%C3%B1ol.pdf

OWASP German Chapter. (Marzo de 2008). *Best Practices: Use of Web Application Firewalls*. Recuperado el 10 de Octubre de 2014, de https://www.owasp.org/index.php/Category:OWASP_Best_Practices:_Use_of_Web_Application_Firewalls

Picazo, R., Villatoro, E., Luna, W., & Jaimez, C. (2014). *Herramienta de apoyo en la detección de reutilización de código fuente.* Journal of Research in Computing Science.

Román, F., Sabido, I., & García, L. (2014). *Capacidades de deteccion de las herramientas de analisis de vulnerabilidades en aplicaciones Web.* Universidad Complutense de Madrid, Grupo de Analisis, Seguridad y Sistemas. Alicante: RECSI.

Salton, G., Wong, A., & Yang, C. S. (1975). *A Vector Space Model for Automatic Indexing.* Cornell University. ACM.

Santillan, M. (25 de Febrero de 2015). *One Million WordPress Websites Vulnerable to SQL Injection Attack*. Recuperado el 25 de Febrero de 2015, de THE STATE OF SECURITY: http://www.tripwire.com/state-of-security/latest-security-news/one-million-wordpress-websites-vulnerable-to-sql-injection-attack/

Scambray, J., Liu, V., & Sima, C. (2010). *Hacking Exposed - Web Applications: Web Applications Security, Secrets and Solutions* (3 ed.). USA: McGraw-Hill.

Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2 ed.). Indianapolis, Indiana, USA: John Wiley & Sons, Inc.

Sunitha, K., & Sridevi, M. (2010). *Automated Detection System for SQL Injection Attack* (Vol. 4). Malaysia: International Journal of Computer Science and Security (IJCSS).

Tf-idf. (09 de Diciembre de 2014). *Wikipedia, La enciclopedia libre.* (L. e. Wikipedia, Editor) Recuperado el 15 de Diciembre de 2014, de Wikipedia, La enciclopedia libre.: http://es.wikipedia.org/w/index.php?title=Tf-idf&oldid=78647781

Torrano, C., Perez, A., & Alvarez, G. (2009). *A Self-learning Anomaly-Based Web Application Firewall.* Burgos: CISIS'09, 2nd International Workshop.

Ulmer, C., & Gokhale, M. (2010). *A Configurable-Hardware Document-Similarity Classifier to Detect Web Attacks*. Sandia National Laboratories and Lawrence Livermore National Laboratory. Atlanta: 2010 IEEE International Symposium on Parallel & Distributed Processing.

UNAM-CERT. (10 de Marzo de 2009). *Aspectos Básicos de la Seguridad en Aplicaciones Web*. Recuperado el 15 de Mayo de 2013, de Coordinación de Seguridad de la Información: http://www.seguridad.unam.mx/documento/?id=17

WASC. (2010). *WASC THREAT CLASSIFICAT ION.* WEB APPLICATION SECURITY CONSORTIUM. Recuperado el 15 de Junio de 2014, de http://projects.webappsec.org/w/file/fetch/13247059/WASC-TC-v2_0.pdf

WASC. (6 de Noviembre de 2011). *The Web Security Glossary*. Recuperado el 26 de Enero de 2014, de The Web Application Security Consortium: http://projects.webappsec.org/w/page/13246967/The%20Web%20Security%20Glossary

Web, W. W. (2015 de Marzo de 4). *Wikipedia, La enciclopedia Libre.* Recuperado el 2015 de Marzo de 10, de http://es.wikipedia.org/w/index.php?title=World_Wide_Web&oldid=80389019