BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE UN PLAN DE AUDITORÍAS DE SEGURIDAD A SISTEMAS WEB BASADOS EN EL FRAMEWORK LARAVEL UTILIZANDO LAS PLATAFORMAS WINDOWS Y LINUX

TESIS

PARA OBTENER EL TÍTULO EN:

LICENCIATURA EN INGENIERIA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

HÉCTOR JOAB IBARRA CABALLERO

ASESOR:

M.C. ANA CLAUDIA ZENTENO VAZQUEZ

Puebla. Pue

Octubre 2019

Dedicatoria.

A mi madre Aurea y a mi padre Héctor que siempre han sido un soporte para mí, son el mejor ejemplo de superación, conocimiento, sabiduría, experiencia y cariño que un hijo puede tener, gracias papás por su amor, apoyo, paciencia, consejos, esfuerzo, trabajo y sacrificio.

A mi hermano Santos que estuvo conmigo cuando nadie más lo hizo y acompañarme en los momentos más difíciles de mi vida, tu apoyo me ayudó a levantarme cuando más lo necesité, gracias por tu amistad hermano.

MC. Ana Claudia por confiar en mi cuando nadie más lo hizo, le agradezco maestra por acompañarme y empujarme para ir un paso más adelante. Gracias por la ayuda y el apoyo que me dio en todo momento, motivarme e inspirarme no solo a terminar la tesis, sino también la carrera profesional. Las palabras siempre me faltarán para expresar lo agradecido que estoy con usted, gracias.

Agradecimientos

A las maestras Carmen, Judith y Yeiny por sus consejos, apoyo y trabajo en esta tesis.

A el maestro Ata por la ayuda que me dio durante la escritura de esta tesis.

A mis ex compañeros de la universidad por acompañarme durante la carrera profesional.

Índice

Resumen 6
Introducción 7
Capítulo 1: Seguridad en la Web 9
1.1 Internet y la ciberseguridad. 9
1.2 Seguridad en los Servicios Web 9
1.3 Ataques a Servicios Web 12
1.4 Ejemplo de Ataques informáticos 16
Capítulo 2: Aplicaciones, Servicios y Seguridad Web 18
2.1 Aplicaciones Web definición, usos y tecnologías. 18
2.2 Servidores Web definición, usos y tecnologías. 19
2.3 Framework 21
2.3.1 Tipos de Framework 22
2.3.2 Modelo, Vista, Controlador 22
2.4 Ataque informático y tipos de ataque. 23
2.5 Vulnerabilidades web más comunes (OWASP) 24
2.6 Plan de auditorías en seguridad 29
2.7 Herramientas en Auditoría y pruebas de penetración 32
Capítulo 3: Laravel Framework 39
3.1. ¿Qué es Laravel Framework? 39
3.2 Sistema de Carpetas de Laravel 39
3.3. Funciones y Métodos de Laravel 42
3.3.1 Artisan 42
3.3.2 Rutas 42
3.3.3 Middleware 42
3.3.4 CSRF 43
3.3.5 Controladores 44
3.3.6 Bycript 44
3.3.7 Modelo/Eloquent ORM 46
3.3.8 Vistas/Blade 47
3.3.9 Encriptado 48
3.4 Desarrollo de un Sitio Web de prueba en Laravel Framework 48
3.4.1 Estructura de la aplicación 50

3.4.2 Rutas 51
3.4.3 Vistas 51
3.4.3.1 Formulario y Vista Registro 52
3.4.3.2 Formulario y Vista Login 53
3.4.3.3 Formulario y Vista Home 54
3.4.4 Base de Datos 54
3.4.5 Controlador 55
3.4.6 Validación 56
3.4.7 Laravel Framework Log Event y error 404 58
Capítulo 4: Desarrollo de un Plan de Auditorías 59
4.1 Plan de Auditoría 59
4.2 Ataques y Pruebas en el Sitio Web 60
4.2.1 Recopilación de información 60
4.2.2 Pruebas de gestión de la configuración y la implementación. 61
4.2.3 Pruebas de gestión de la identidad 62
4.2.4 Pruebas de autenticación 63
4.2.5 Pruebas de autorización 63
4.2.6 Pruebas de gestión de sesiones seguras en Laravel 64
4.2.7 Pruebas de validación de entrada a Bases de Datos con CO2 Plugin 65
4.2.8 Control de errores – Acceso a ruta no autorizadas de la aplicación 66
4.2.9 Criptografía 67
4.2.10 Pruebas de lógica empresarial 68
4.2.11 Pruebas del lado del cliente 70
4.3 Resultados 72
4.3.1 Resultado de ZAP 72
4.3.2 Resultados de auditoría con Burp Suite 76
Capítulo 5: Conclusiones 78
5.1 Plan de Auditoría en Seguridad Preventivo 80
5.2 Trabajo Futuro 85
Anexo: Manual de instalación de Laravel Framework en Ubuntu 18 y Debian 8. 86
Anexo: Manual de Instalación de Kali Linux en VMWare. 91
Listado de Ilustraciones. 97
Bibliografía 99

Resumen

La evaluación de vulnerabilidades de las aplicaciones web que se encuentran en la red y utilizan las herramientas basadas en la tecnología en seguridad propuestas por un Framework del lado del servidor es una constante en el mundo de la seguridad web. Existen organismos internacionales que documentan y elaboran estadísticas de las fallas de seguridad reportadas todos los días por los especialistas de la seguridad, con el fin de dar un panorama actualizado de seguridad. En este trabajo de tesis se diseñó un plan de auditoría con el cual se establece un proceso de seguridad que puede dar seguimiento a las fallas de una aplicación web y por consiguiente evaluar las vulnerabilidades encontradas, con el fin de documentar para su posterior corrección. Para detectar vulnerabilidades, se propone diseñar un sitio con un Framework que ofrezca seguridad al producto de software. Laravel es un Framework del lado del servidor, que implementa un paradigma de desarrollo de aplicaciones web elegante y seguro basado en el lenguaje de programación PHP. En este trabajo de tesis, se explica el proceso de desarrollo y documentación de una aplicación web en la cual se emplean las herramientas, métodos y funciones de seguridad de Laravel Framework, usando como base un sistema operativo Debian basado en GNU/Linux montado en una máquina virtual. Para trabajar la evaluación de vulnerabilidades se documentaron una serie de pruebas de penetración utilizando las herramientas en análisis de vulnerabilidades OWASP Zed Attack Proxy Project ZAP y Burp Suite instaladas en el sistema operativo Kali Linux, con el objetivo validar que las herramientas de Laravel Framework cumplen con lo establecido en su documentación. Finalmente se propone la implementación de un plan de auditorías, donde se enlistan 5 puntos que se deben de cubrir desde el inicio del proceso de desarrollo de una aplicación web para asegurar los productos que se desarrollan cuentan con la menor cantidad de vulnerabilidades durante el desarrollo, implementación y mantenimiento.

Introducción

Internet se ha convertido en la más importante herramienta de los últimos años, solo en México hay 71 millones de personas con acceso a la red. Dentro de la red se puede encontrar una infinidad de sitios de entretenimiento, servicios bancarios, tramites gubernamentales, compras en línea, redes sociales y correo electrónico, con todos los servicios que se pueden ofrecer en la red es común que las empresas implementen de los servicios en una aplicación web para distintas funciones o tareas propias de la empresa. Para poder lograr esto, los desarrolladores web utilizan diversas herramientas de desarrollo para crear aplicaciones que sean atractivas, fáciles de usar y que realicen de manera segura la función para la que fueron desarrolladas, el uso de distintas tecnologías de desarrollo como los Framework facilitan estos procesos de desarrollo. Un Framework cuenta con el potencial de, no solo mejorar el aspecto visual de una aplicación web, sino también de facilitar el esfuerzo de programación implementando nuevos paradigmas de desarrollo utilizando librerías, método y funciones incluidos por el Framework. Con todos los avances en el área del desarrollo web es normal confiar en que las herramientas de desarrollo son seguras. Por ejemplo, al usar una aplicación web es común que realicemos un proceso de registro y validación de cuenta de correo electrónico, en algunos casos vinculamos las cuentas de Facebook, Google o alguno otro servicio web para poder tener acceso a una página web. Pero, ¿Realmente es segura la aplicación web que estamos utilizando?, un usuario de una aplicación web no puede solucionar o protegerse completamente de los peligros que pueden aparecer durante su navegación en una página web. Los equipos profesionales de desarrollo no han sido capaces de encontrar todos esos eventos que supone un riesgo para los usuarios, así como para el sistema. En los últimos años las noticias hablan de fugas de información de grandes compañías como Facebook

o Google, ataques informáticos a distintos servicios web, con el objetivo de obtener una base de datos de usuarios y contraseñas.

Aunque las fugas de información suelen pasar por desapercibido para muchos usuarios, el daño real comienza cuando estas bases de datos sirven para realizar otro tipo de ataques a un sistema bancario, alguna empresa o sitio de gobierno, suplantación de identidad, robo de cuentas bancarias etc. lo que lleva a preguntarnos. ¿Puede un plan de auditoría en seguridad y análisis de vulnerabilidades implementado durante los procesos de desarrollo, implementación y mantenimiento de un sistema evitar las fallas en seguridad de una aplicación web? Las constantes pruebas de penetración y documentación de vulnerabilidades pueden ayudar a solucionar estas fallas de raíz, pero para entender todos estos procesos implementados en un plan de auditoría en seguridad es necesario desglosar cada uno de estos temas para finalmente culminar con una propuesta para mejorar la seguridad del sistema que resguarda los datos de los usuarios.

Capítulo 1: Seguridad en la Web

1.1Internet y la ciberseguridad.

Internet es una herramienta que ha revolucionado el mundo durante los últimos años, las distintas organizaciones, instituciones de gobierno, financieras, de servicios, etc., hacen uso de la red para recibir, enviar y almacenar información. La información que se envía, incluye la identidad en la red y los datos de un usuario, esto es uno de los recursos informáticos más importantes que se pueden encontrar en la red, ya que incluyen información personal, como historiales médicos, registros financieros en línea, etc. Por esta razón es que constantemente se busca proteger esta información del robo. Pero ¿A dónde va nuestra información al enviarla por la red?, ¿Quiénes tienen almacenada esta información? y lo más importante conocer si los servicios web a los que damos nuestra información como usuarios cumplen con las políticas de no divulgación y cuentan con la seguridad necesaria para proteger el robo de información de los usuarios. Se puede definir la ciberseguridad como *el esfuerzo constante por proteger estos sistemas de red y todos los datos contra el uso no autorizado o los daños* [1], Los servicios en línea deben de ofrecer la confidencialidad, integridad y disponibilidad de la identidad de un usuario en la red y sus credenciales para evitar ser objetivo de los delincuentes en internet. Las consecuencias de no cumplir con estos requisitos pueden llevar a una organización, empresa o usuario a ser víctima de vandalismo, robo, suplantación de identidad, entre otras amenazas en internet.

1.2Seguridad en los Servicios Web

Dentro de internet podemos encontrar diversas aplicaciones web que brindan servicios como tiendas en línea, bancos, oficinas de gobierno y sistemas de gestión para empresas. Estos sistemas envían y reciben los datos de un usuario constantemente y siempre existen riesgos al manejar esta

información, desde la forma en como interactúa con otros sistemas o el mismo servidor. Esto lleva a que las compañías trabajen en solucionar las fallas de seguridad de los sistemas digitales implementado parches de software a sus sistemas. Este tipo de prácticas es considerado por los especialistas de la seguridad como un práctica ineficaz, ya que, como se describe en la siguiente ilustración 1, existe una ventana de vulnerabilidad, definida como el tiempo de reacción que existe para los atacantes de explotar una falla en seguridad dentro del sistema, estos tiempos van desde que se descubre la falla, se reporta, se notifica al usuario, generar un análisis de la falla y crear el parche, la posterior liberación del parche para finalizar con la instalación [2].

Ilustración 1: Ventana de Vulnerabilidad

Muchas de las vulnerabilidades críticas y de alto riesgo se pueden encontrar dentro de las líneas de código de una aplicación web, las malas prácticas de programación llevan a tener sistemas con

un alto riesgo de vulnerabilidad, aunque nuestro servidor se encuentre protegido por una serie de distintas capas de seguridad.

Se he documentado que las fallas de seguridad más comunes se encuentran en las herramientas que interactúan con el usuario, en específico los servidores web, utilizan una mala configuración o por defecto de un servidor, ya que facilita el posible acceso de agentes externos. Es común escuchar noticias de compañías que sufrieron ataques informáticos debido al desconocimiento de actualizar el software que ocupan para trabajar, ya sea de un servidor, programas de uso cotidiano, incluso de librerías, compiladores y el propio Framework de desarrollo. Existen herramientas que permiten interrumpir y capturar información durante los procesos de transferencia, por esta razón la comunicación cliente-servidor siempre debe de trabajar con métodos de cifrado, aunque muchos de estos canales de cifrado trabajen todavía con métodos anticuados como MD5, SSLv2 y SSLv3. El uso de métodos de cifrado puede evitar caer en vulnerabilidades. La captura de información, el uso de cuentas con contraseñas débiles o sin contraseña es una falla muy común, las fugas de información sobre usuarios como Google y Microsoft, han permitido que las bibliotecas de contraseñas se hayan incrementado, ya que son víctima de algoritmos de fuerza bruta que usan bibliotecas de datos y esta problemática ha tomado relevancia en los últimos años.

Una de las tantas maneras que se tiene para proteger los sistemas es el uso de respaldos y el registro de eventos, los respaldos nos permiten tener puntos donde nuestros sistemas trabaja sin errores y en caso de necesitarlos podremos restablecer completamente un sistema dañado en cuestión de minutos. Mientras que el registro de eventos nos permite tener un panorama general de lo que sucede en el servidor y alerta de posibles fallas, incluso de actividades extrañas en el sistema.

Los servicios web suelen recibir miles de peticiones de clientes. Un analista de seguridad debe de tener en cuenta que cada cliente es un posible atacante, por lo que agregar medidas de acceso a

nuestro servidor puede ser uno de los mejores métodos para evitar la acumulación de peticiones de acceso de dudosa procedencia (IP falsas), incluso tener una gran cantidad de puertos abiertos representa un riesgo latente en un servidor.

1.3Ataques a Servicios Web

Es posible conocer los riesgos más comunes que corre una aplicación considerando el número de vulnerabilidades reportadas y documentadas por grupos como Sucuri en su reporte Sucuri Hacked Report 2017 y Open Web Application Security Project (OWASP) en el OWASP Top 10 The Ten Most Critical Web Application Security Risks-2017, a las que llegan fallas reportadas para cualquier aplicación, independiente de la plataforma, arquitectura o sistema operativo. OWASP nos muestra una evaluación de las amenazas dentro de un sistema en la ilustración 2. Un agente o atacante puede usar diferentes formas de penetrar dentro de un sistema, estos caminos representan el riesgo que puede ser o no, tan grave para llamar nuestra atención.

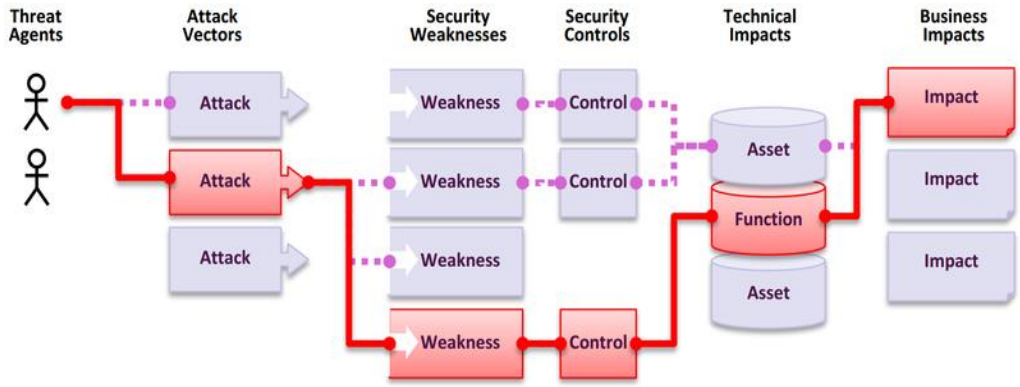
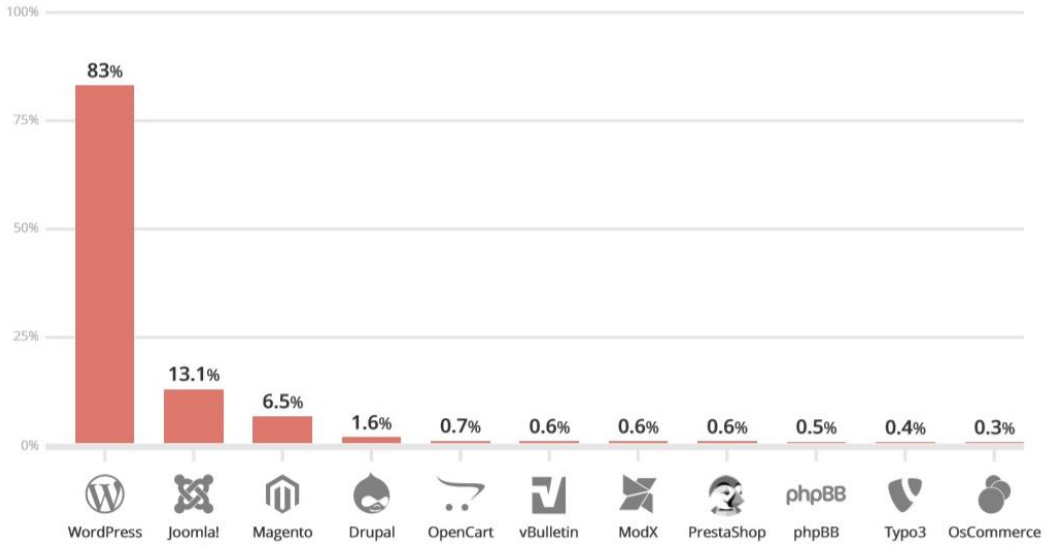


Ilustración 2: Evaluación de una amenaza; OWASP Top 10-2017[3]

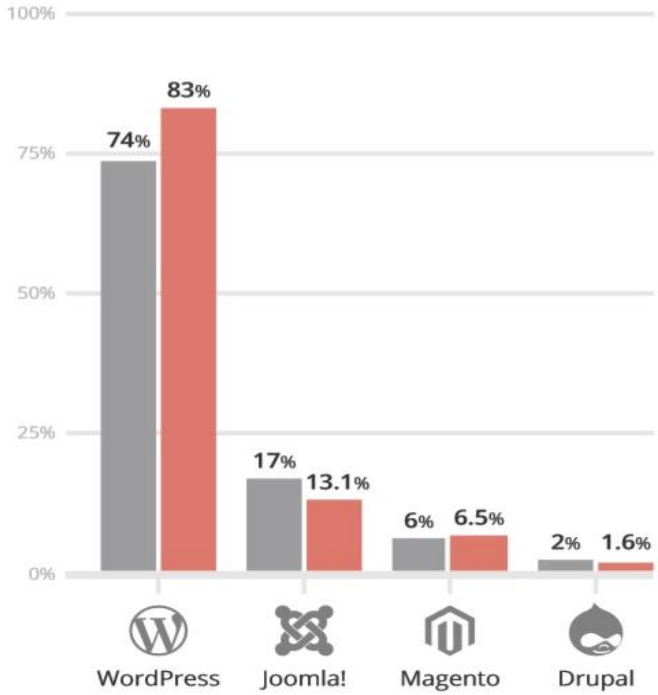
Estos caminos pueden ser fáciles de explotar o extremadamente difíciles de llevar a cabo. Para esto se puede evaluar la probabilidad asociada con cada amenaza, vector de ataque y debilidad de seguridad, combinarla con una estimación del impacto técnico y comercial en su organización estos factores determinan un riesgo real.

Sucuri y OWASP recabaron los reportes de fallos de los Content Management System (CMS) que fueron comprometidos y se encuentran registrados en sus respectivas website blacklist. En la gráfica de plataformas web infectadas Sucuri, como en la gráfica 1 y 2 de plataformas web infectadas OWASP, WordPress, Joomla, Drupal y Magento, son las plataformas más afectadas durante el 2017.

Gráfica 1: Plataformas Web Infectadas; Sucuri 2017[4]



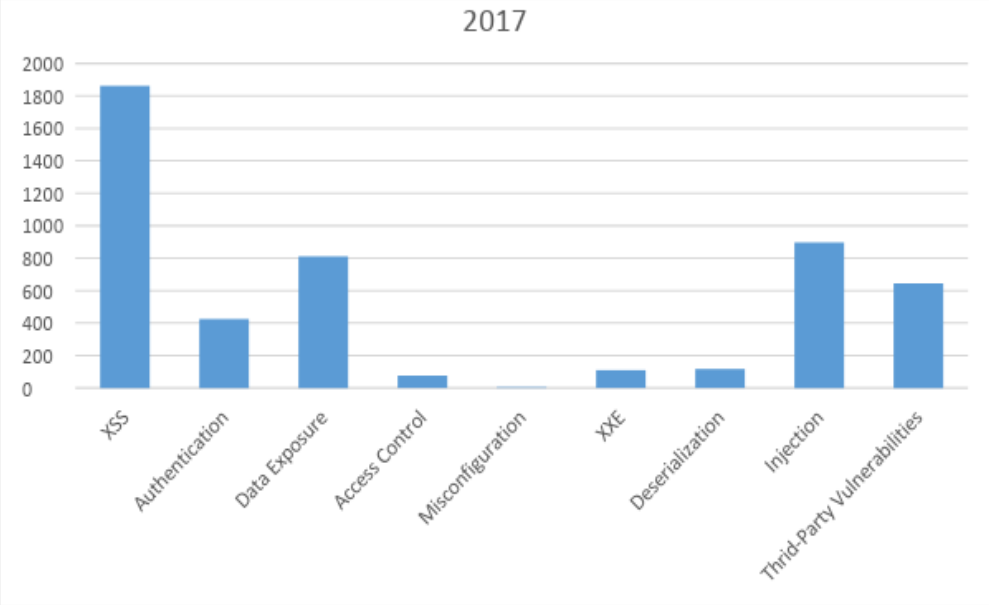
Gráfica 2: Plataformas Web Infectadas; Sucuri 2017[5]



Durante este análisis se encontró un incremento en las vulnerabilidades dentro de contenedores de aplicaciones web. WordPress se ha convertido en el objetivo de cientos de hackers, ya que son muy conocidas sus fallas en seguridad y aunque cientos de sitios web sobre seguridad recomiendan el uso de Drupal, este aun cuenta con fallas dentro de su sistema de seguridad.

Consultando datos recabados durante 2017 por OWASP, más del 50% de las aplicaciones web tienen una vulnerabilidad pública visible para los piratas informáticos, 36% de estas vulnerabilidades no cuentan con una solución, ya sea por medio de una actualización o un parche de versión.

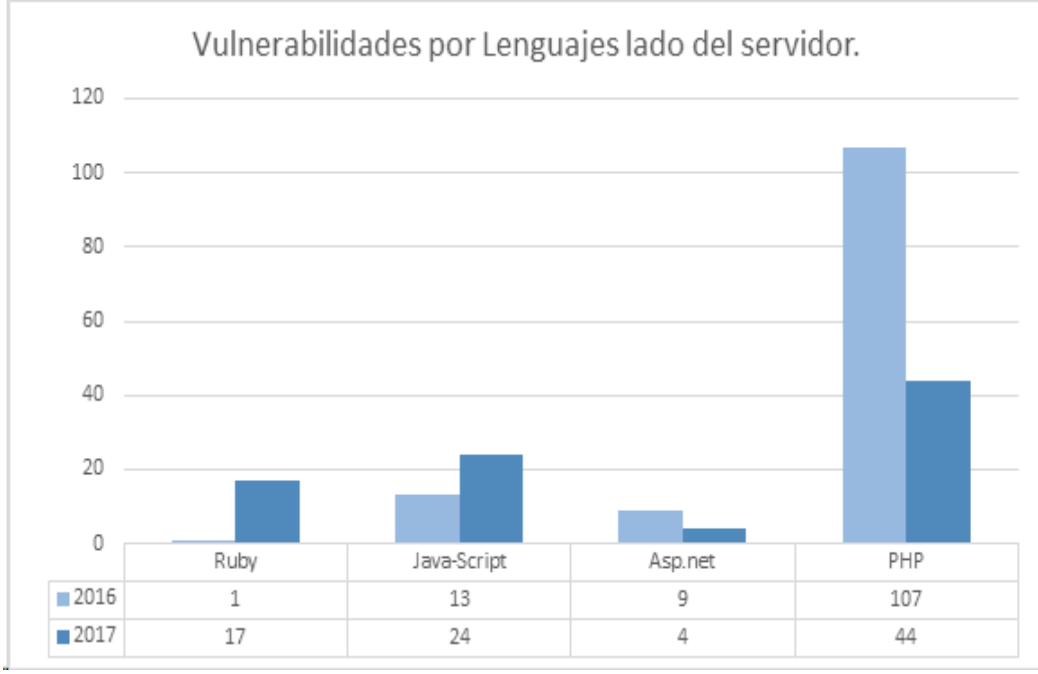
Gráfica 3: Principales amenazas recabadas por número de incidentes: OWASP 2017[5]



En la gráfica 3 se muestra que, de las principales vulnerabilidades, sobresalen los ataques como Cross-Site Scripting (XSS), Injection, Data Exposure, Third-Party Vulnerabilities and Authentication, convirtiéndose en las principales amenazas que deben de ser cubiertas por las aplicaciones web y los servicios que las almacenan.

Dentro de los lenguajes del lado del servidor, Common Vulnerabilities and Exposures recaba la información de las vulnerabilidades encontradas dentro de estos lenguajes con se observa en la gráfica 4, PHP sigue siendo el más usado, pero con altos niveles de vulnerabilidades, pero menores a las vulnerabilidades encontradas durante el 2016 [6].

Gráfica 4: Vulnerabilidades en lenguajes; CVE 2018 [6]



1.4Ejemplo de Ataques informáticos

Durante el 2017 y 2018 se han presentado diversas noticias de ataques a grandes plataformas como Facebook y Google+. Facebook anunció el 25 de septiembre que se había encontrado una vulnerabilidad en una de sus funciones "ver cómo", la cual permite guardar información de login (token de acceso) para evitar que el usuario tenga que ingresar su información para poder tener acceso a su cuenta. En marzo del mismo año durante el caso de Facebook con Cambridge Analytica, Google+ anunció un fallo, el cual permitía obtener información a desarrolladores de aplicaciones de terceros, accediendo a datos marcados como privados (dirección de correo, ocupación, género, edad, etc.).

En un reporte de seguridad en aplicaciones web publicado por Verizon Data Breach Report 2017 [7] habla que 3 de cada 10 brechas de seguridad tuvieron como origen ataques a aplicaciones web, la tasa de brechas de seguridad derivadas de vulnerabilidades en las web apps ha aumentado un 300% entre 2014 y 2016. Uno de los ejemplos más grandes que tenemos de lo grave que puede ser

una pequeña vulnerabilidad en un sistema es Equifax, esa brecha de seguridad dejó más de 147 millones de clientes expuestos, este error provino de no actualizar el framework de desarrollo de aplicaciones web Apache Struts. El 99% de las veces los ataques suceden por cosas comunes y sencillas: sistemas que no fueron parcheados, contraseñas débiles, malware en un endpoint,por lo que, el riesgo de seguridad ha pasado de la red hacia la capa de aplicaciones y los endpoints.

Datos publicados por Imperva [8], muestra que las vulnerabilidades de Cross-Site scripting XSS representan el mayor número de vulnerabilidades de aplicaciones web en 2018. Según las predicciones de Imperva continuarán siendo las ofensivas más frecuentes en 2018. La inserción SQL se encuentran en el segundo lugar como uno de los peligros más recurrentes, SQL se usa frecuentemente para la administración de cuentas de usuarios, lo que se convierte en un objetivo de los cibercriminales, buscando obtener información valiosa.

Una de las mejores formas de evitar ataques a una aplicación web interna es el desarrollo de una aplicación segura, evitar Frameworks débiles y en caso de usar uno actualizar sus librerías cada vez que sus desarrolladores liberan una nueva versión. Evitar enviar información no codificada a través de funciones en JavaScript y métodos GET.

En México se han presentado diferentes ataques al Sistema de Pagos Electrónicos Interbancarios (SPEI), según el reportaje realizado por la revista proceso, Banxico ha elevado sus niveles de alerta de seguridad informática en las operaciones de los sistemas de pago, tras un ataque a la aseguradora AXA el 26 octubre del 2018 [9]. Mientras que en abril y mayo de 2018 se reveló que 836 cuentas bancarias de 10 institución diferentes fueron objeto de fraude durante una intervención cibernética vulnerando los sistemas de transferencias electrónicas. Este es un ejemplo de los continuos ataques que han sufrido las transferencias bancarias electrónicas a los bancos mexicanos y es una cifra que va en aumento en el cierre del año 2018.

Capítulo 2: Aplicaciones, Servicios y Seguridad Web

2.1 Aplicaciones Web definición, usos y tecnologías.

Las páginas o aplicaciones web son documentos digitales creados a partir de lenguajes de programación, pueden contener texto e imágenes, reproducir sonido y video, capacidad de ejecutar programas, enlaces, etc. Se encuentran contenidas dentro de la World Wide Web (WWW) y se pueden acceder a estas por medio de un navegador web. Las aplicaciones web son utilizadas en la actualidad en diferentes categorías.

- De gestión interna este tipo de aplicaciones tienen la función de trabajar para generar facturación, contabilidad, registro y administración de usuarios.
- Herramientas de trabajo como intranets, trabajo de red. Comunicación ya sean para enviar correo, chats etc.
- Herramientas Web para tiendas virtuales especializadas, webs complejas con diferentes elementos de entrada y salida, repositorio o buscadores.
- También tenemos de varios usos ya sean páginas de mapas, plataformas educativas, entretenimiento, etc.

Con todas las ventajas que ofrece una aplicación web es normal que sean usadas por tiendas, empresas, oficinas gubernamentales, universidades etc., encontrar aplicaciones dedicadas a trabajar como tiendas digitales en las que muestran en catálogos miles de productos y que permiten hacer pedidos directos a los productores a través de sus formularios de venta. Aplicaciones web que nacieron para agilizar trámites gubernamentales, manejo de cuentas bancarias, etc., con estos ejemplos básicos de los muchos a los que se le pueden dar una aplicación web, se debe tener en

cuenta que los millones de usuarios y sus datos personales, se mueven en la red constantemente en un flujo de información sensible, lo que propone retos constantes a la seguridad web.

2.2 Servidores Web definición, usos y tecnologías.

Un servidor web es un sistema programado para procesar conexiones unidireccionales o bidireccionales, síncronas o asíncronas que permite escuchar y responder peticiones de un cliente. Los servidores web se han convertido en una herramienta imprescindible en la actualidad, ya que almacenan la información que se encuentre en la red, así como también los servicios a los que nos conectamos todos los días, pero cuya función principal es la interacción entre un cliente y a la máquina, para esto el sistema tiene que poder gestionar de manera eficiente los recursos de memoria, archivos y procesos de la computadora.

Un Sistema Operativo (SO) puede ser usado como servidor, pero solo podrá proporcionar algunos servicios básicos, ejecutar aplicaciones y programas multiusuario, comunicarse con otros servidores, atender solicitudes específicas de un cliente. Tenemos dos formas de clasificar los servidores, dedicados y no dedicados.

A. Los dedicados son aquellos que funcionan exclusivamente para un solo usuario y estos pueden ser adecuados a las necesidades del usuario ya sea hardware y software. El cliente puede disponer completamente de todos los recursos del servidor sin tener que compartir los recursos del servidor con algún otro proceso de distinto usuario, en los servidores dedicados se puede mejorar la gestión de los recursos del sistema y los procesos en ejecución para mejorar el rendimiento de las aplicaciones ejecutadas para el servidor, el mantenimiento, así como las actualizaciones pueden realizarse con mayor facilidad y control. Dentro de los servidores dedicados podemos distinguir de dos tipos de servidores:

1. Dedicados administrados
2. Dedicados no administrados

La principal diferencia que se puede encontrar en estos dos servidores es la gestión del servidor, un usuario de un servidor dedicado administrado puede dejar la gestión del sistema desatendido para que un externo se encargue solamente del software y hardware de la máquina, mientras que en un servidor dedicado no administrado el usuario es completamente responsable de la actualización del sistema, funcionamiento del software y hardware de la máquina.

B. En un servidor no dedicado tenemos varios usuarios compartiendo todos los recursos de la máquina, esto abarata sus costos de uso, pero también limita su rendimiento. Podemos encontrar distintos tipos de Sistemas Operativos los cuales pueden ser habilitados para trabajar específicamente como servidores web como Windows Server y las distintas distribuciones basadas en los kernel de Linux.

Un Framework del lado del servidor es un conjunto de herramientas y bibliotecas que se ejecutan en el servidor web, para simplificar operaciones del desarrollo de aplicación web, conexiones a base de datos, protocolos de cifrado y seguridad, operaciones internas que no son visibles para el cliente. Uno de los Frameworks del lado del servidor más populares y desarrollado por Microsoft es *.NET Framework*, que propone un independiente y rápido desarrollo de aplicaciones, buscando implementar un sistema que se integre fácilmente a sus productos, nacido para competir contra el creciente desarrollo de aplicaciones basadas en JAVA de Oracle Corporation, así como de los múltiples Frameworks basados en PHP.

Gracias a su compatibilidad de desarrollo y un propio sistema, JAVA entró al mercado con muchos de los mejores Frameworks para el desarrollo web basado en JavaScript, uno de ellos y con una

amplia popularidad actualmente es Angular desarrollado en base a *TypeScript*, un lenguaje de código abierto mantenido por Microsoft, siendo este, un súper conjunto de JavaScript. Principalmente es utilizado para crear aplicaciones web de una sólo página y así darles a los usuarios una sensación de un sistema fluido y con una carga dinámica, que reacciona a las acciones de los usuarios agregando los elementos solicitados por este, haciendo de esta manera que el desarrollo, como las pruebas de los sistemas sean más fáciles.

Laravel es propuesto como uno de los mejores Frameworks de código abierto, basados en PHP para el desarrollo de aplicaciones web, cuenta con una gran variedad de métodos y funciones que garantizan su utilidad como la compatibilidad con otros Frameworks. Cuenta con su propio entorno de trabajo conocido como *Laravel Homestead* pero con la posibilidad de ser compatible con otros sistemas que pueden ser Linux, Windows o Mac OS. Laravel es sin duda una de las mejores formas de iniciar en el moderno desarrollo web profesional y gracias a su compatibilidad se pueden aportar mejoras en la seguridad de este Framework cuando está almacenado en servidores basados en Linux.

2.3 Framework

Un Framework es un conjunto de componentes en software (librerías, compiladores, intérpretes y API) que conforman un entorno de trabajo. Cuenta con modelos, prácticas y conceptos los cuales buscan hacer más rápido y fácil el desarrollo de aplicaciones web. Actualmente los Frameworks se han convertido en herramientas de programación útiles para dedicarse de forma exclusiva en la programación de un lenguaje o varios. La mayor parte de los Frameworks se enfocan en el desarrollo de aplicaciones web, la gestión de usuario, manejo de datos, control de plantillas o rutas, etc., por lo que trabajan perfectamente ofreciendo servicios web.

2.3.1 Tipos de Framework

Los Framework pueden ser divididos en dos tipos dada la forma en la que interactúan con las aplicaciones web, Front-End y Back-End, como se muestra en la ilustración 3 un usuario que interactúa con una aplicación web, la cual contiene en su totalidad código HTML y en su defecto algún Front-End Framework, solo observa las funciones básicas de la aplicación y que incluirán las implementaciones visuales añadidas por el Front-End Framework, algunas de las cuales pueden ser elementos visuales como botones, casillas, marcos, otras de la cuales pueden ser mallas para organizar los elementos estéticos de las aplicaciones y funciones para tener aplicaciones responsivas. Dentro de nuestro Servidor Web, se encuentran almacenados los archivos HTML, los mismos archivos con los que un usuario interactúa, podemos encontrar un Back-End Framework el cual trabaja del lado del servidor, agregando métodos, funciones o simplemente facilitando el trabajo del servidor junto con la aplicación web, conexiones a bases de datos e implementando filtros de seguridad para los archivos y datos.

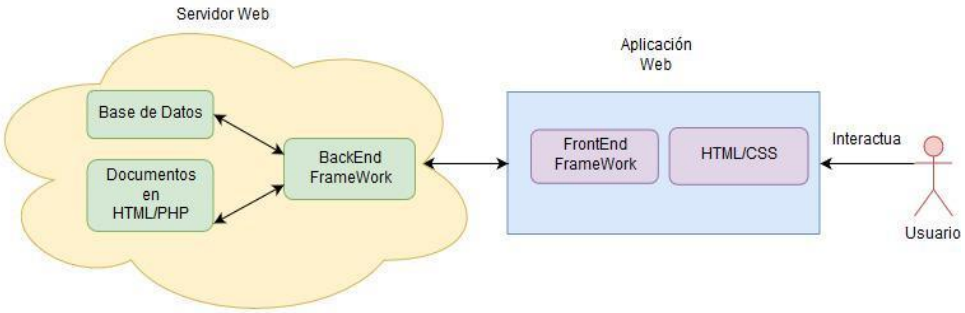


Ilustración 3: Interacción de un Framework Back-End y Front-End

2.3.2 Modelo, Vista, Controlador

Una de las principales características empleadas por los Back-End Framework es el Modelo Vista Controlador (Model View Controller, MVC), ejemplificado en la ilustración 4, el cual consiste en mantener separadas las funciones básicas de una aplicación, el modelo, la vista y el controlador,

estos tres componentes interactúan de forma que solo dejan ver al usuario una parte de su interacción con la aplicación.

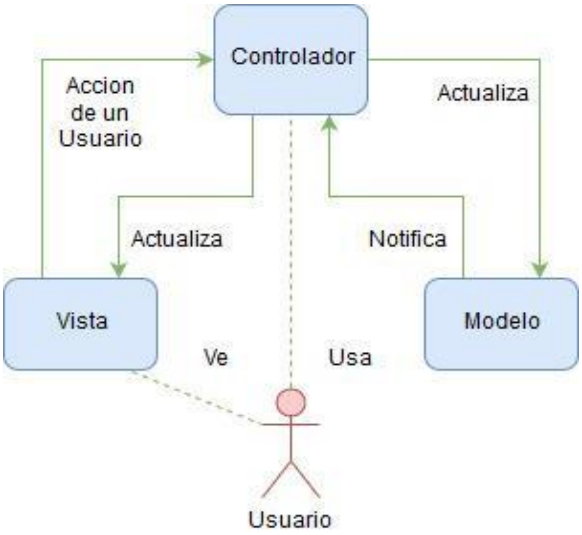


Ilustración 4: Modelo, Vista, Controlador. MVC

2.4 Ataque informático y tipos de ataque.

Seguridad Web esté definido como el conjunto de procedimientos, prácticas y herramientas tecnológicas para la protección web, servidores, usuarios y organizaciones. Seguridad que nos protege de inesperados comportamientos en la red [10]. Un ataque cibernético es un acto intencional por parte de un usuario malicioso con el fin de aprovecharse de una vulnerabilidad en el diseño o desarrollo de una aplicación, ejemplificado en la Ilustración 5.

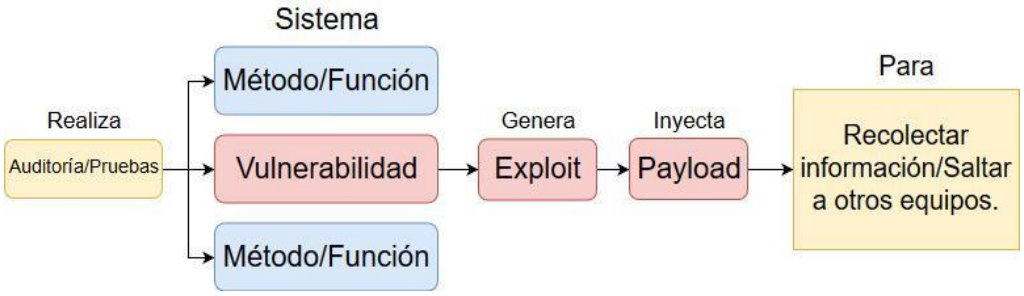


Ilustración 5: Diagrama de un ataque cibernético

Después de realizar diferentes pruebas de penetración/auditorias y al encontrar una vulnerabilidad en un sistema, se genera un Exploit. Un Exploit es la codificación de distintas reglas que aprovechan la vulnerabilidad encontrada para obtener un acceso al sistema, generalmente se dice que los Exploit son llaves para abrir estas puertas de acceso encontradas durante las pruebas o evaluación de la seguridad de un sistema y no el código malicioso que infecta un sistema.

Existen cientos de miles de Exploits identificados, agregados a las firmas de seguridad de distintas compañías y organizaciones dedicadas a la seguridad informática. Pero como se puede intuir, cada día se evalúa la seguridad de distintos sistemas y al encontrar una puerta de acceso, esto permite generar un Exploit para aprovechar esa vulnerabilidad. Estos Exploit son conocidos como 0-Day el mismo día que se encuentra son implementados o vendidos en el mercado negro sin notificar la falla a los responsables. Por lo que es imposible tener un conocimiento exacto de la cantidad de Exploit existentes como para tener una seguridad completa.

Después de terminar la ejecución de un Exploit y abrir una puerta dentro del sistema lo que procede a trabajar con los *Payload*, paquetes de datos que se introducción al sistema deseado y tomar control del sistema, muchos de estos Payload incluyen una terminal (Shell) dentro del sistema externo permitiendo recolectar información, así como saltar a otros equipos dentro de la misma red.

2.5 Vulnerabilidades web más comunes (OWASP)

En la actualidad existen cientos de organizaciones dedicadas a manejar y evaluar la seguridad web, recabando información de diversos reportes que van surgiendo alrededor del mundo, muchas de estas amenazas son identificadas y documentadas por OWASP. Open Web Application Security

Project creó un documento sobre los más grandes riesgos en seguridad de las aplicaciones Top 10 Most Critical Web Application Security Risks [3].

El Top 10 de riesgos a la seguridad nos muestra que están asociados a un número de agentes maliciosos, vector de ataque, vulneración de seguridad, un combinado del impacto técnico y administrativo a las organizaciones y juntos todos estos factores nos dan un numero de riesgo general, que a continuación se describen.

A1: 2017 - Injection: Es un ataque de infiltración de código utilizando las vulnerabilidades de administración a la base de datos de alguna aplicación web. Las vulnerabilidades de inyección se encuentran en las consultas a SQL, LDAP, XPath o NoSQL, comandos de sistema operativo, analizadores XML y encabezados SMTP. Estas fallas son fáciles de encontrar si se utilizan escáneres o fuzzers (Fuzzer o Fuzzing: Tecnica de pen-testing programable, que consiste en el envió de entradas aleatorias a una aplicación). Los ataques de inyección pueden resultar en pérdidas de información, corrupción de datos, denegación de acceso, así como pérdida de control en la administración de la base de datos. Para evitar estos en necesario validar cada uno de los datos proporcionados por los usuarios.

A2:2017 - Broken Authentication: Un atacante puede detectar fallas en la autentificación de forma manual o teniendo acceso a una lista de cientos de millones de nombres de usuarios y contraseñas, cuentas de administrador por default y ataques de fuerza bruta. El impacto realizado por un ataque así puede comprometer todo el sistema, permitiendo lavado de dinero, robo de identidad, fraude en seguridad social y liberación la información sensible o protegida. Este tipo de vulnerabilidades pueden ser evitadas implementando autenticación de múltiples factores para evitar ataques automatizados de relleno de credenciales como fuerza bruta, no utilizar credenciales

por default, verificar las contraseñas débiles y cambiarlas, aplicar medidas de limitación de intentos de login.

A3:2017 - Sensitive Data Exposure: Cuando manejamos información sensible como contraseñas, números de tarjetas de crédito, información médica o personal, información secreta de empresas y dependencias de gobierno, esta información se convierte en un objetivo y puede ser tomada a través de ataques de criptoanálisis, robo de llaves de cifrado, ataques tipo man-in-the-middle o ataques manuales de fuerza bruta utilizando bases de datos millones de contraseñas. En los últimos años se ha vuelto el ataque más común, por el simple hecho de que no se emplean herramientas de encriptado o las herramientas de encriptado, usan llaves débiles o por defecto. Una falla de este tipo puede comprometer toda la información almacenada, para evitar esto es necesario clasificar la información con la que se está tratando, aplicar controles para cada clasificación, no almacenar información sensible dentro de variables débiles o temporales, utilizar sistemas de encriptado, protocolos y llaves para nuestra información sensible, la transmisión se debe realizar de acuerdo con protocolos como TLS (Transport Layer Security) con cifrados PFS (Perfect Forward Secrecy) y almacenar las contraseñas en funciones de hash.

A4:2017 - XML External Entities (XXE): De las diversas formas que tenemos para vulnerar un sistema se encuentran las entidades externas XML, son documentos XML que incluyen código hostil para nuestro sistema, estos pueden ser enviados de forma predeterminada. Para evitar esto es necesario estudiar la forma de identificar los XXE, como no usar formato de datos complejos como JSON, actualizar los procesadores y librerías XML, así como SOAP, deshabilitar las entidades externas XML y los procesadores DTD en los analizadores de aplicaciones. La implementación de un “whitelist” del lado del servidor para toda validación de entrada nos ayudará a prevenir datos hostiles provenientes de documentos XML, headers o nodos.

A5:2017 - Broken Access Control: Es común encontrar debilidades en los controles de accesos de una aplicación, ya que la mayoría de los desarrolladores no cuentan con la detección automatizada o manuales de pruebas de control de acceso. Los impactos de este tipo de ataques van desde la creación, acceso, descarga y eliminación de toda información dentro de nuestra aplicación, El control de accesos puede mejorarse si se trabaja con código del lado del servidor o server-less API, un atacante no puede modificar verificadores de control de acceso o metadata.

A6:2017 - Security Misconfiguration: Las malas configuración de seguridad se pueden encontrar en distintas partes de una aplicación web, los servicios de red, plataforma, servidor web, base de datos, marcos, código personalizado, máquinas virtuales, contenedores o almacenamiento. Estos errores pueden ser evitados si se utilizan escáneres automatizados, ya que estos detectan las fallas que se tienen dentro de la configuración, ya sea por permisos de cuentas de usuario, configuraciones predeterminadas o servicios innecesarios.

A7:2017 - Cross-Site Scripting (XSS): Es una vulnerabilidad informática o agujero de seguridad común en las aplicaciones web, este tipo de vulnerabilidad permite un atacante inyectar en páginas web código JavaScript o algún otro lenguaje similar, algo que se puede evitar usando medidas CSP (Política del mismo origen), existen diferentes Frameworks que pueden ayudar a evitar esto, ya que usan marcos que escapan de manera automática a los XSS.

A8:2017 - Insecure Deserialization: Este puede ser considerado uno de los peligrosos y raros tipos de ataques que se pueden realizar a una aplicación web o API, estos ataques están relacionados con la desconfiguración de los objetos y la estructura de datos de los paquetes internos de una aplicación web, buscando modificar la lógica de la aplicación para lograr la ejecución de código remoto.

A9:2017 - Using Components with Known Vulnerabilities: Actualmente las aplicaciones web se ven beneficiadas con el nacimiento de diferentes tecnologías dedicadas a complementar y aumentar sus capacidades incluyendo librerías o paquetes de archivos para mejorar su funcionamiento, casos como el de Apache Struts 2 que contaba con una vulnerabilidad en su código, por el cual varias aplicaciones pudieron ser vulneradas. Es un problema cuando se arreglan estos errores a través de parches o actualizaciones de las librerías añadidas a la aplicación, por lo que es necesario remover dependencias sin usar, herramientas, componentes, documentos y documentación innecesarias, verificar las versiones de los componentes Client-Side y Server-Side, así como obtener estos componentes en sus sitios oficiales de preferencia, con firma digital, actualizar los componentes y evitar las versiones anteriores si poseen vulnerabilidades.

A10:2017 - Insufficiente Logging & Monitoring: Son errores producidos por tener un login insuficiente, detección y seguimiento, cuando los mensajes de error o advertencias no existen, son inadecuadas o confusos, cuando un login falla o no ocurre como se espera, se producen vulnerabilidades que un atacante puede aprovechar para capturar información sensible, para evitar estos riesgos en seguridad es necesario verificar el funcionamiento de fallo de inicio de sesión, control de acceso y validación de entrada proporcionado por el lado del servidor, así como comprobar que exista un adecuado registro con la información de los usuarios para identificar cuentas sospechosas, tener un registro de accesos que permanezca un tiempo suficiente en caso de necesitar un análisis forense, estos registros deben de ser generados fácilmente, manejar controles de integridad para evitar la manipulación o eliminación. Un monitoreo de actividades sospechosas puede ayudar a detectar y responder ataques de forma rápida, existen planes de recuperación de incidentes, en los que se puede trabajar.

2.6 Plan de auditorías en seguridad

Las distintas empresas dedicadas al desarrollo de programas en seguridad (Avast, McAfee, Eset, etc), proponen el uso de herramientas tecnológicas para atender y evitar la explotación de las fallas en seguridad, la desventaja de usar estas herramientas es que no atienden de forma concreta la raíz de una vulnerabilidad, solo ofrecen una capa de seguridad basada en firmas y bases de datos en programas sospechosos. Organizaciones internacionales con OWASP y Offensive Security OSCP, recomiendan el uso de planes de auditoría en seguridad para poder evaluar un sistema de forma más completa. Un plan de auditorías en seguridad es una serie de pasos a seguir cuyo fin es la evaluación de vulnerabilidades y la creación de un reporte sobre el estado del sistema. La mayor parte de las Testing Guide que se encuentran en la red suelen ser muy generales, cumbre un amplio espectro de posibles fallas en el sistema, pero al tratar con sistemas más complejos o que implementan tecnologías más específicas suelen fallar debido a que ningún sistema es igual a otro. Es necesario entender que la seguridad es un proceso por lo que este tipo de metodologías deben de ser creadas e implementados siguiendo un estándar. Estos estándares son propuestos por los equipos de pentesters internos o externos, basándose en información actualizada de las últimas y más comunes fallas en seguridad. Es necesario conocer las herramientas en software usadas durante el desarrollo del sistema web, así como de la documentación generada a partir de inicio del desarrollo del sistema hasta su liberación, esto para poder dar un seguimiento del ciclo de vida del sistema de manera que se pueda construir un panorama donde se podrían encontrar fallas y enfocar los esfuerzos en secciones más específicas. Una vez que ya se creado un mapa general de la composición del sistema a evaluar, es necesario crear una lista de cuáles son los puntos que el plan de auditoría debe de cubrir para asegurar que un sistema es seguro, por ejemplo:

- Recopilación de información

- Gestión de configuración e implementación
- Gestión de identidad
- Autenticación
- Autorización
- Gestión de sesiones
- Validación de entrada
- Manejo de error
- Criptografía
- Lógica de negocios
- Lado del cliente

Finalmente se podrán proponer una serie de pruebas para cubrir cada uno de estos puntos, así como las herramientas que se usarán durante el plan de auditorías propuesto, todos los planes de auditoría implementan pruebas de caja: Black Box Testing, White Box Testing y Gray Box Testing.

Black Box Testing es un método de evaluación estándar donde se desconoce la estructura interna del sistema, se asumen dos posiciones, el de un usuario estándar y la de un agente malicioso. Como usuario estándar se realizan distintas pruebas al sistema de manera manual usando el navegador, proporcionando entradas y salidas al sistema, en la que se evalúan las respuestas de este, validaciones, notificaciones de errores y mensajes de ayuda del sistema. Como agente malicioso es necesario poseer un conocimiento en cómo debe de comportarse una aplicación web para poder explotar esas posibles vulnerabilidades, se prueban entradas que normalmente un usuario estándar no implementaría (Consultas SQL, JavaScript, peticiones Get y Post), con el fin de observar si el sistema responde de manera anormal.

White Box Testing es un método de evaluación avanzado en pruebas de penetración en el cual la estructura interna, el diseño y la implementación de un sistema son evaluados a detalle para documentar y registrar los niveles de riesgo del sistema. Es necesario que un pentester o evaluador

con conocimientos avanzado en informática utilizando herramientas en pruebas de penetración implemente ejercicios de entrada y salida a las diferentes funciones lógicas del sistema, para verificar que cada una de las respuestas del sistema están siendo validadas y se comporten como lo esperado. Normalmente estas pruebas requieren la información de cómo trabaja el sistema interno a evaluar, el sistema de red, versiones y herramientas de software, así como del código de la aplicación, estructura de la base de datos, etc.

Grey Box Testing es una combinación de las pruebas de caja negra y caja blanca, por lo que involucra implementar ejercicios como un agente malicioso y una vez que se encuentra una vulnerabilidad escalar a las pruebas para la explotación y la generación de un reporte de la vulnerabilidad encontrada. Aunque dentro de estas pruebas se tenga un conocimiento limitado del código de la aplicación web, no involucra su alteración para pruebas de mayor rendimiento con las pruebas de caja blanca.

Los equipos de pentesters normalmente trabajan con pruebas de caja negra y caja gris durante sus planes de auditorías, generan un reporte y notifican a los responsables del sistema los resultados obtenidos. Mientras que internamente los equipos de desarrollo trabajan con pruebas de caja blanca donde evalúan sus sistemas y aplicaciones. El propósito de documentar estas pruebas, es el crear un proceso comparativo del estado del sistema o aplicación a cierto criterio de evaluación. Comúnmente este criterio de evaluación y el riesgo interno de las vulnerabilidades encontradas no está definido o establecido, suele variar en un estado de cambio en la percepción del evaluador, por lo que es recomendable proponer una escala comparativa de resultados basados en los registros de OWASP los 10 riesgos de seguridad más críticos en aplicaciones web. Por lo que el plan de auditoría comprende la evaluación del sistema, la aplicación de las pruebas, documentación de las pruebas y la creación de un reporte de riego.

2.7 Herramientas en Auditoría y pruebas de penetración

Dentro de las distintas herramientas para realizar auditorías y evaluación de la seguridad informática en un sistema, cabe destacar el uso de distintos sistemas operativos diseñados para realizar estas tareas Parrot Security OS, BackTrack Linux, BackBox, en este trabajo de tesis se trabajó con una distribución Kali Linux, es un avanzado sistema operativo en pruebas de penetración y auditoría basado en el kernel de Debian, diseñado y mantenido por Offensive Security OSCP, salió con su primer versión estable el 13 de Marzo de 2013 como una reconstrucción completa del antiguo sistema en seguridad BackTrack Linux donde se eliminaron herramientas obsoletas así como otras herramientas que realizaban la misma función, evitando programas repetidos dentro del sistema aun así cuenta con más de 600 herramientas prueba. Este sistema operativo es completamente gratuito y de código abierto para poder ser configurado a necesidades específicas. Kali Linux cuenta con un Filesystem Hierarchy Standard (FHS), lo que permite al SO poder encontrar binarios, archivos de soporte, librerías, etc. Uno de los principales sectores que recibe servicio es el soporte para dispositivos Wireless, lo que permite tener una compatibilidad con diversos dispositivos Wireless, incluidos los USB [11].

Dentro de Kali Linux se pueden encontrar distintas herramientas de trabajo ordenadas según su función, como recolección de información, análisis de vulnerabilidades, ataques Wireless, scan para aplicaciones web, herramientas de explotación, pruebas de estrés, herramientas forenses, sniffers y spoofing, ataques a password, mantenimiento de accesos, ingeniería inversa, hardware hacking y herramientas de reporte. Kali Linux es un SO muy completo diseñado específicamente para la auditoría, cumple con normas y pruebas para uso profesional, por lo que su equipo de desarrollo consta de un pequeño grupo el cual al modificar sus paquetes oficiales y añaden una firma por desarrollador (GNU Privacy Guard). Una de las muchas ventajas que nos permite

trabajar con Kali Linux es la posibilidad de trabajar en distintas arquitecturas como i386. Amd64 y ARM, esto habla de que Kali Linux puede trabajar correctamente en cualquier computadora, así como en un dispositivo Android o una tarjeta Raspberry Pi [11].

De todas las herramientas con las que cuenta Kali Linux se explicar a continuacion dos OWASP Zed Attack Proxy Project (ZAP) y Burp Suite, son herramientas para el análisis de aplicaciones web que trabajan como un proxy junto con el navegador. Estos dos Framework en seguridad trabajan con pruebas pasivas y activas, vinculando sus respuestas con el compilado de información en riesgo de OWASP, lo que permite tener pruebas de evaluación de vulnerabilidades en base a normas de una organización internacional.

Una prueba activa es un escáner integrado dentro de algunos programas que evalúan las vulnerabilidades en una aplicación web, también pueden generar reportes en base a la información obtenida comparándola con los listados de riesgos y vulnerabilidades vinculados a la aplicación. Estos reportes muestran información de donde ha localizado la vulnerabilidad, muestra consejos de cómo evitarlas y qué daño pueden causar. Como se muestra en la ilustración 6, en estos escáneres se implementan diversos scripts, programas, algoritmos de vulneración de contraseñas, fuzzing, validación de protocolos, certificados, etc. Las herramientas usadas dentro de estas pruebas activas trabajan en forma de módulos, que pueden ser activados y desactivados dentro del programa, lo que permite agregar nuevos módulos de pruebas no enlistadas dentro del escáner si es que se encuentran disponibles para su aplicación. Lo que permite realizar pruebas de forma más rápida y tener un panorama general de la situación de riesgo en la que se encuentra nuestro Sistema.

La evaluación en un escáner que nos ayudan a realizar pruebas de forma manual, permitiendo evaluar sectores específicos de una aplicación, por lo que se proponen diferentes escenarios a los que se puede enfrentar la seguridad dentro de una aplicación. Estas pruebas pasivas realizadas a

una aplicación proporcionan información detallada y se pueden encontrar vulnerabilidades no enlistadas dentro de las respuestas con las que se programa un escáner, pero llevan más tiempo y recursos que una prueba activa. Analizar los procedimientos que se realizan durante estas pruebas facilitan comprender el funcionamiento de la aplicación web que se evalúa, la repetición en prueba y error es una de las mejores formas de auditoría que se pueden realizar, ya que se validan en base a los conocimientos y la experiencia que se tiene de las vulnerabilidades en ese justo instante. También ayudan al evaluador a poder presentar un reporte de evaluación más completo y con una mayor exactitud al calificar un riesgo dentro de un Sistema.

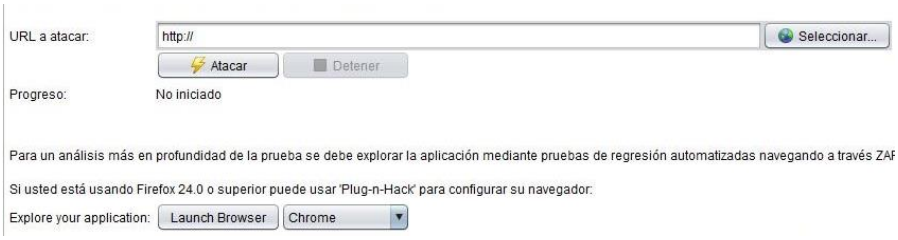


Ilustración 6: Zed Attack Proxy Project_consola de pruebas

OWASP Zed Attack Proxy Project o ZAP es un Framework de auditoría multiplataforma, flexible y escalable, puede trabajar para la detección de vulnerabilidades con pruebas pasivas y activas, por lo que se considera un proxy de intercepción (Man-in-the-middle proxy), ilustración 8 o Stand-Alone como un proceso daemon, ilustración 7.

Ilustración 7: Stand-Alone [12]

Ilustración 8: Intercepting Proxy [12]

ZAP es Open Source, por lo se puede abrir el código del Framework para añadir nuevas herramientas, así como para arreglar errores y trabajar de forma libre mejorando el sistema. ZAP es gratuito en su versión completa, por lo que se pueden encontrar distintos tutoriales, cursos y documentación que sirven como un apoyo a las personas iniciadas en el área de auditoría y pruebas de penetración en aplicaciones web.

Cuenta con certificados SSL y anti token handling CSRF, un escáner automático de sistema que utiliza herramientas como Spider para el mapeo del sitio, un fuzzer que permite inyectar de forma automática datos random al sistema y estudiar su comportamiento. ZAP también cuenta con pantallas de alerta donde se muestran los resultados obtenidos de nuestra evaluación de vulnerabilidades, dentro de estas pantallas se describe la vulnerabilidad, se explica donde se encontró, cómo se podría explotar y una recomendación de cómo evitarlo.

A diferencia de ZAP, Burp Suite es una herramienta comercial para la evaluación de vulnerabilidades, así como también pruebas de penetración, cuenta con tres versiones, su versión gratuita Community, la Professional con un costo de 399 Dólares y la Enterprise de 3999 Dólares, costo de licencia por un año. La versión Burp Suite Community usada en este proyecto permite a los evaluadores la posibilidad de interceptar todas las peticiones y respuestas entre la aplicación objetivo y el buscador, incluso cuando el protocolo HTTPS es usado. Se puede ver, editar y enviar mensajes individuales para manipular los componentes Server-Side y Client-Side dentro de la aplicación.

Como se muestra en la ilustración 9, el proxy de Burp Suite permite mantener un historial de todos los detalles de las peticiones y respuestas capturadas, permitiendo al evaluador detallar ítems individuales con comentarios o resaltar con colores para ayudar al evaluador.

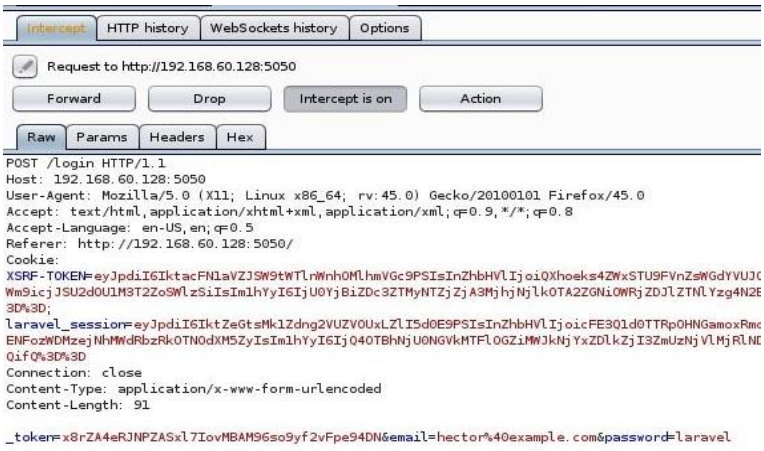


Ilustración 9: Consola del proxy vista, edición y envío de mensajes de Burp Suite

Se pueden generar una modificación de respuesta automáticas para facilitar la evaluación, por ejemplo, se pueden ver campos de formulario escondidos, habilitar campos del formulario inhabilitados y remover código JavaScript para validación.

Con las herramientas de emparejamiento y reemplazo de reglas para que esta modificación de respuestas automáticas pueda pasar a través del proxy, estas reglas pueden ser creadas dentro de los mensajes en los encabezados y el cuerpo de la aplicación web, al solicitar parámetros o en el URL de la aplicación, como se muestra en la ilustración 10.

Ilustración 10: Emparejamiento y reemplazo de reglas [13].

Burp Suite ayuda eliminando las alertas de seguridad en el navegador cuando existe una intercepción de conexiones HTTPS, esto es porque Burp Suite añade una autoridad de certificación (CA) que podemos instalar en el navegador, estos certificados son generados en cada dominio que se visita.

Burp Suite cuenta con una sección conocida como Mapa del sitio objetivo, donde se muestra todo el contenido que ha sido descubierto por el evaluador. Este contenido se muestra en forma de un árbol jerárquico correspondiente a la estructura del URL. Al seleccionar estos nodos se muestra cada uno de los ítems individuales escuchados con detalle de las peticiones y respuesta disponibles como podemos observar en la ilustración 11.

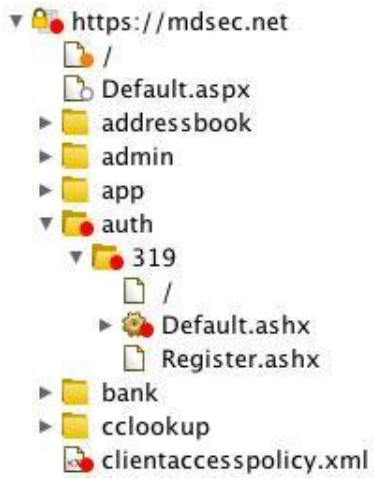


Ilustración 11: Mapa del sitio objetivo [13].

La regla de interacción permite tener un control preciso de los mensajes que son interceptados, un filtro para destacar las interacciones más interesantes de la aplicación. Todas la peticiones y respuestas se muestran en un editor de mensajes HTTP, este provee numerosas vistas en mensajes subyacentes para ayudar a analizar y modificar su contenido.

Todas estas peticiones y respuestas pueden ser enviadas a las distintas herramientas con las que cuenta Burp Suite, en la ilustración 12 podemos visualizar todas estas herramientas.

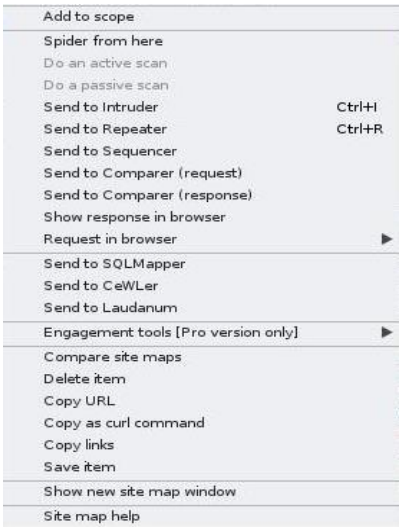


Ilustración 12: Envío de respuestas y peticiones

Burp Suite cuenta con herramientas que ayudan a la evaluación de vulnerabilidades y pruebas de penetración, en la ilustracion 13 podemos ver una de las más notables, la BApp Store un repositorio de extensiones listos para usar, desarrollados por la comunidad de usuario de Burp. Complementos que podrán ayudar a realizar tareas más específicas dentro de las pruebas de un evaluador.

Ilustración 13: BApp Store [13].

Capítulo 3: Laravel Framework

3.1. ¿Qué es Laravel Framework?

Laravel es un Framework multiplataforma basado en las versiones de PHP 5 y 7, cuyo fin es el desarrollo de código PHP de forma ordenada, simple y elegante. Laravel incluye dentro de sus paquetes, manejadores de dependencias Symfony y Composer lo que complementa la capacidad en el desarrollo de aplicaciones y servidores en PHP. Laravel es un Framework completo, desarrollado para satisfacer las demandas y necesidades del desarrollo de aplicaciones web modernas basadas en la estructura MVC. Actualmente se encuentra en su versión 5.8, pero se trabajara en la versión 5.5 debido a su estabilidad.

3.2 Sistema de Carpetas de Laravel

Laravel Framework, es una colección de distintos métodos, funciones y herramientas, los cuales trabajan entre sí para generar buenas prácticas de desarrollo. Propone estas ventajas en forma de un Framework de código abierto, el desarrollador puede acceder a cualquiera de los archivos de Laravel y adecuarlo a sus necesidades, así como de dar la posibilidad de crear nuevas funciones y métodos.

Una de las principales propuestas de Laravel para el desarrollo de aplicaciones, es el uso de “rutas con cierres”, donde se incluye el uso del clásico desarrolló bajo la arquitectura de software MVC, pero utilizando rutas, con las que se responderán las peticiones HTTP que provengan de algún usuario. De esta manera al acceder a una ruta del servidor envía el correspondiente tipo de petición a una función dentro del controlador para que responda al usuario con la vista deseada. En caso de que haya una petición a la base de datos, ya sea para ingresar, buscar, modificar o eliminar información, el controlador pedirá al modelo que realice la operación asignada sobre la base de

datos y este confirme al controlador sobre los resultados obtenidos y en caso de ser necesario el controlador notifica al usuario o mostrar los cambios por medio de una vista como se observa en la ilustración 14.

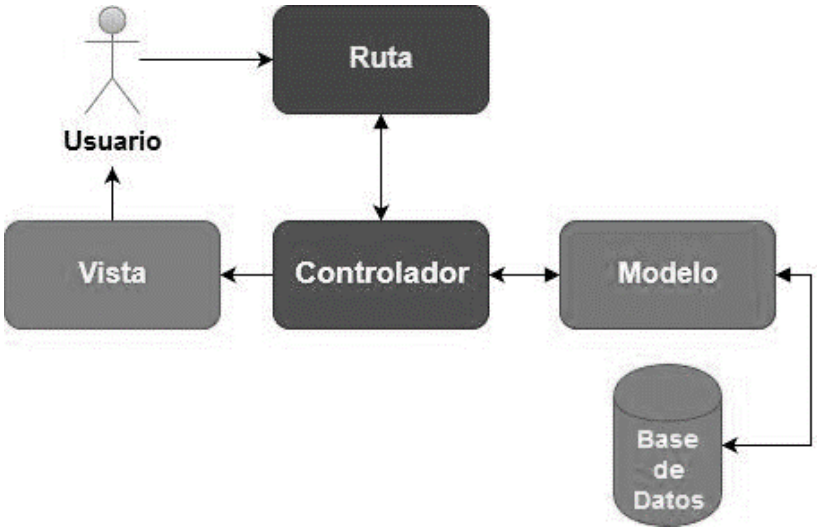


Ilustración 14: Estructura RMVC en Laravel

Al crear un proyecto en Laravel este muestra la estructura de directorios explicado en la ilustración 15. Estos directorios son separados en dos principales estructuras, directorios Root y directorios APP. En el directorio de root se puede encontrar un directorio de Bootstrap, en el archivo app los scripts de arranque de Laravel, así como un directorio cache donde se generan los archivos de performance y optimización, tambien el caché de ruta y servicios. En el directorio Config como su nombre lo indica, se encuentra la configuración del framework y de la aplicación que se desarrolla. Dentro de Database se encuentran los archivos relacionados con el manejo de la base de datos, Migrates, Seeder y Eloquent. El directorio Public contiene los recursos estáticos de nuestra aplicación, archivos, css, js, imágenes y fuentes. Resources contiene tres subdirectorios.

1) assets (ubica los archivos less de la aplicación).
2) lang (archivos de internacionalización para pasar el idioma del proyecto de uno a otro).

3) views (nuestras vistas en formato PHP o php:blade. Además se agrega una carpeta templates para las plantillas).

Uno de los directorios más importante es Routes, que almacena archivos relacionados con el enrutamiento de Laravel. RouteServiceProvider, es un archivo ubicado en el grupo del middleware proveyendo de estados de sesión, protección CSRF, encriptamiento de cookies, autenticación vía token, etc. Cuando Laravel necesita guardar archivos de sesión, cache, templates o diversos archivos generados por el framework lo hace a través del directorio Storage. Vendor es un directorio donde se encuentran todas las dependencias de composer. Y finalmente en el directorio Tests donde los archivos de pruebas son ejecutados posteriormente por phpunit. Dentro de los directorios Root podemos encontrar App donde se podrá encontrar el núcleo de la aplicación. Incluye las clases de la aplicación, archivos de configuración, etc. Dentro se encuentran otros subdirectorios como consola, donde podemos encontrar los comandos de la consola de Artisan, events, exceptions siendo uno de los más importantes HTTP, donde se encuentran los controllers, middlewares y requests.

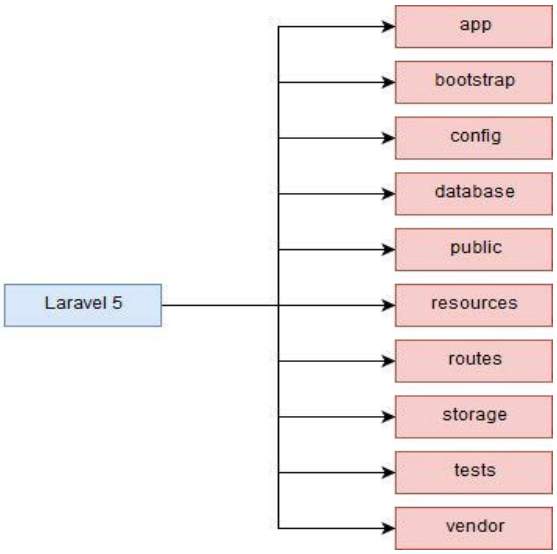


Ilustración 15: Estructura interna de las carpetas de Laravel

3.3. Funciones y Métodos de Laravel

Dentro de todas las funciones y herramientas que contiene Laravel Framework es necesario profundizar en algunas características para poder entender el objetivo del proyecto y como se usara más adelante.

3.3.1 Artisan

Laravel Framework cuenta con una interfaz en línea de comandos, que brinda asistencia para poder construir nuestra aplicación, estos comandos pueden ser llamados usando una terminal dentro de la carpeta del proyecto. Se puede consultar una lista de 16 etiquetas con el comando php artisan list. Si se quiere ver los argumentos disponibles, así como las opciones para cada una de estas etiquetas podemos agregar php artisan help ###, donde ### es la etiqueta deseada, podemos ver en el ejemplo de la ilustracion 16.

```
demon@2806-10a6-0008-273a-020c-29ff-fe7e-4986:/var/www/html/mipagina$ php artisan db:se
ed
Seeding: UserSeeder
```

Ilustración 16: Ejemplo de comando artisan en consola.

3.3.2 Rutas

Laravel Framework trabaja con un sistema de rutas con cierre, esto quiere decir que cada vez que se accede a una ruta del sistema solo se podrá acceder por una puerta y estas serán definidas en nuestro archivo de rutas, estas rutas son asignadas al grupo del middleware, lo que nos provee de un estado de sesión, así como protecciones Cross-site Request Forgery(CSRF).

3.3.3 Middleware

Dentro de Laravel Framework existe un mecanismo de filtrado para petición HTTP para una aplicación. El middleware verifica que un usuario dentro de la aplicación esté autentificado y en

caso de no serlo procederá a llevarlo a la pantalla de login. El middleware provee más funciones, como un Cross-origin resource sharing(CORS), responsable de añadir los debidos headers para las respuestas de salida de la aplicación, un logging middleware registra las solicitudes de entrada a la aplicación, etc.

3.3.4 CSRF

Antes de comenzar se define el concepto de CSRF, por sus siglas en inglés Cross-Site Request Forgery; falsificación de petición en sitios cruzados. Es un tipo te Exploit malicioso de un sitio web en el que comandos no autorizados son transmitidos por un tercero.

¿Cómo Laravel nos protege te estos tipos de ataque? Las versiones de Laravel 5.0 en adelante habilitan el middleware VerifyCsrToken, que por defecto genera un Token por cada usuario del sistema y este Token es el que utiliza el middleware para verificar que cada una de las peticiones sea legítima. Pero no analiza las peticiones 'HEAT', 'GET' y 'OPTIONS'.

Este Token funciona de la siguiente manera, cada vez que se crea una ruta de tipo POST utilizando ROUTE o llamando el método POST, se agrega dentro de un formulario un llamado al método csrf_filed(), como se observa en la ilustración 17.

```
<form method="POST" action="{{ url('usuarios/crear') }}">
    {{ csrf_field() }}

    <button type="submit">Crear usuario</button>
</form>
```

Ilustración 17: Campo de Token CSRF

En la ilustración 18 se observa como se agrega un campo oculto llamado _token con el valor del Token al código HTML del formulario el cual verificará que cada una de las peticiones sea legítima de un usuario. Revisando nuestra clase VerifyCsrToken se puede observar cómo es que está

compuesto nuestro método con el cual podemos agregar URLs para ser excluidas de la verificación y puedan ser usadas en algunos casos.

Ilustración 18: Middleware

3.3.5 Controladores

Los controladores funcionan recibiendo las peticiones de las rutas, por medio del envió de una vista o la modificación de algún sector dentro del sistema. Se pueden incluir funciones que permitan trabajar directamente con el middleware permitiendo la validación de información añadida al sistema, así como de mantener sesiones de usuario para el acceso a otras rutas definidas dentro del sistema. La función dentro del controlador podemos añadir métodos para verificar cada uno de los valores recibidos por un formulario, esto nos brinda la posibilidad de agregar distintos filtros a la información que se recibe del usuario. Una de las distintas herramientas para el tratamiento de información en el controlador es el uso de Bycript, que a continuacion se describe.

3.3.6 Bycript

Hash es una función computable mediante un algoritmo, como se describe en la ilustración 19 al recibir una cadena de entrada procede a crear una nueva salida de tamaño finito normalmente cadenas.

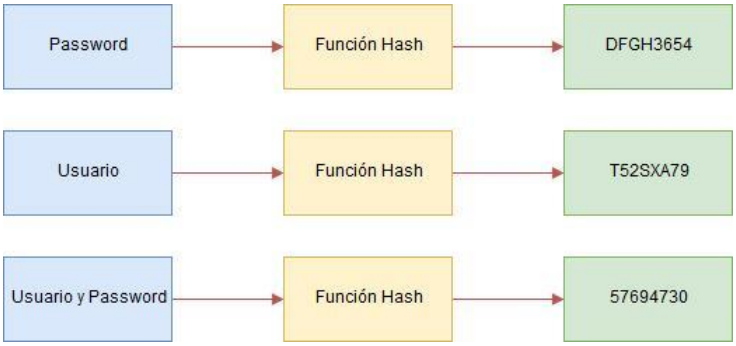


Ilustración 19: Ejemplo de cifrado Hash

Laravel Hash Facade provee un seguro Bcrypt y Argon2 hashing para almacenar las contraseñas de los usuarios. Si se usa un constructor de clase como LoginController y RegisterController que Laravel incluye en la aplicación, estas clases usan Bcrypt para el registro y autentificación por defecto, como se muestra en la ilustración 20, la aplicación de la funcion hash para una contraseña de un usuario.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use App\Http\Controllers\Controller;

class UpdatePasswordController extends Controller
{
    /**
     * Update the password for the user.
     *
     * @param  Request  $request
     * @return Response
     */
    public function update(Request $request)
    {
        // Validate the new password length...

        $request->user()->fill([
            'password' => Hash::make($request->newPassword)
        ])->save();
    }
}
```

Ilustración 20: Cifrado Hash en Laravel

El controlador hash de nuestra aplicación se encuentra configurado en “config/hashing.php” y es apoyado de Bcrypt y Argon2, como podemos observar en la ilustración 21.

Ilustración 21: Archivo de la función Bycript en Laravel

3.3.7 Modelo/Eloquent ORM

Entre las distintas características que podemos encontrar, la administración de la base de datos utilizando Laravel Framework es una de las más complejas, como podemos observar en la ilustración 22, se pueden crear bases de datos con tablas y definir cada una de las características dentro de los campos, todo esto usando Migraciones y Semillas, compiladas gracias a la consola Artisan de Laravel. Pero la función más importante dentro del manejo de base de datos, es el uso de Eloquent ORM, es un patron de arquitectura (Active Record) implementado para el trabajo de base de datos. Nos permite trabajar con consultas tipo query en las tablas de la base de datos, insertando, modificando y eliminando información.

Los métodos predefinidos de Eloquent, agregan una protección para tratar con la inyección SQL, por lo que es posible agregarlos manualmente en métodos construidos por el usuario.

Ilustración 22:Archivo de la función Eloquent ORM en Laravel

3.3.8 Vistas/Blade

Al trabajar con el RMVC, la creación de vistas se mantiene separado del resto de componentes, podemos encontrar nuestras vistas creadas en la carpeta resources/views, estas pueden ser llamadas ya sea desde las rutas o el controlador, dependiendo del tipo de retorno que se necesite realizar. Para el manejo eficiente de las vistas y evitar código repetido, Laravel nos provee del uso de un manejador de plantillas conocido como Blade que permite el uso de etiquetas para el manejo de layout y el uso de etiquetas cortas en php para el manejo de información segura implementando el uso de doble corchete {{$ejemplo}}.

Este tipo de etiquetas convierten los datos recibidos en cadenas de texto, evitando que el código JavaScript sea ejecutado dentro de nuestro sistema. Esta funcion corresponde a una protección al XSS añadido dentro de todas las variables tratadas con Blade.

3.3.9 Encriptado

Laravel utiliza Open SSL dándo un encriptado AES-256 y AES-128, todos los valores encriptados de Laravel son registrados con un código de autenticación de mensaje (MAC) para que su valor no se pueda modificar una vez encriptado.

Para que podamos usar el encriptado de Laravel tenemos que usar la opción "key" que se encuentra dentro de "config/app.php". El comando "php artisan key generate" genera una key con un generador random de bytes en PHP para construirla, ver ilustración 23.

Ilustración 23: Generación de la llave en Laravel

Los valores encriptados se serializan, esto permite que objetos y matrices puedan ser cifrados. Los clientes que no estén utilizando PHP y reciban estos valores cifrados deben serializar los datos. En caso de que nosotros no queramos cifrar y descifrar los valores sin serializar se pueden utilizar otros métodos como encryptString y decryptString de la facade Crypt de PHP.

$encrypted = Crypt : : encryptString('Hello World.');

$decrypted = Crypt : : decryptString('$encrypted');

3.4 Desarrollo de un Sitio Web de prueba en Laravel Framework

Laravel cuenta con su propio entorno de trabajo conocido como Laravel Homestead y Valet para sistemas Linux y MAC el cual satisface los requisitos para poder analizar la seguridad y el entorno de trabajo. Se propuso una instalación limpia del Framework utilizando tecnologías basadas en Linux, se instaló en una máquina virtual una distribución basada en Linux (Debian 8), donde se

agregaron los componentes (Apache 2, MySQL, PHP 7.2) necesarios para el desarrollo de la aplicación web y del Framework Laravel (Composer, Artisan). Para entender mejor la construcción del entorno de trabajo y su estructura, se anexa la ilustración 24, desarrollado a continuación.

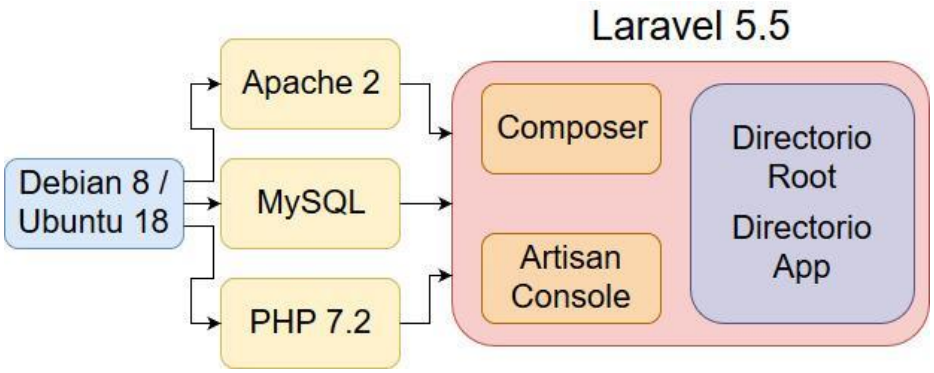


Ilustración 24: Integración de la plataforma, componentes y software.

Debían 8: La distribución seleccionada para la implementación del sistema es Debian 8, ya que es compatible con la mayor parte de paquetes y recibe actualizaciones constantemente servirá como prueba de nuestro Sistema.

Composer: Es una manejador de dependencias y paquetes PHP, necesario para Laravel.

Apache: Es un servidor web HTTP, para plataformas GNU/Linux, Microsoft, Macintosh y contiene la base de lo que se mostrará en el navegador web una vez que Laravel esté instalado.

MySQL/MariaDB: No es un requisito para el funcionamiento de Laravel, pero si de un Servidor LAMP y también es incluido dentro del propio entorno de trabajo Laravel HomeStead como en muchos del sistema desarrollados utilizando PHP es necesario una base de datos y cualquiera de estos dos cubren perfectamente con esa función.

PHP 7.2: Es el lenguaje de programación base para Laravel 5.5 ya que es una de los requisitos para su funcionamiento no cambiaremos el lenguaje de programación sin embargo en un servidor LAMP se pueden usar otros lenguajes como Perl y Python.

Laravel 5.5 LT: Es una versión estable y que aún recibe mantenimiento, aunque actualmente la última versión de Laravel sea la 5.8.

3.4.1 Estructura de la aplicación

La mayoría de los servicios web en línea trabajan con un sistema de Entrada y Registro dentro de sus estructuras principales, por lo que durante el desarrollo del proyecto se crearon dos vistas básicas de Entrada y Registro, donde se usaron las diferentes herramientas proporcionadas por Laravel Framework para realizar una aplicación segura. Se añadieron funciones para validar la información recibida, con el objetivo de evitar la mayor cantidad de vulnerabilidades dentro de una aplicación. Para realizar esta aplicación se siguió las recomendaciones de desarrollo propuestas por Laravel Framework siguiendo el esquema RMVC mostrado en la ilustración 25, por lo que la creación de cada una de las secciones de la aplicación se encuentra en su respectivo sistema de carpetas y serán descritas a continuación.

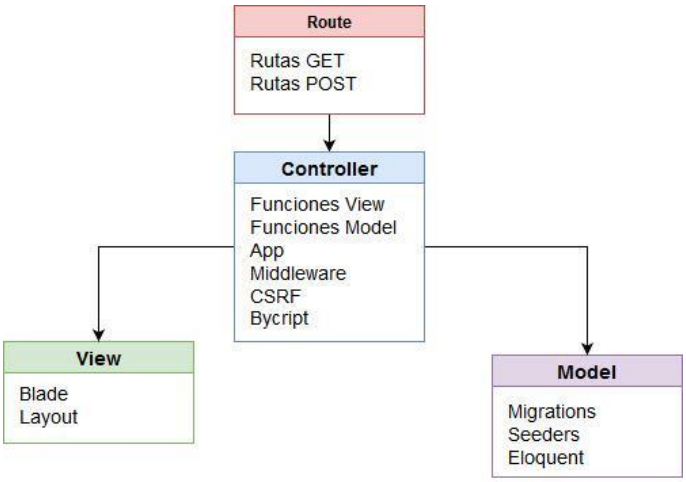


Ilustración 25: RMVC de la aplicación web creada

3.4.2 Rutas

Las rutas disponibles del sistema fueron creadas de Routes y declaradas en web.php, Como se muestra en la ilustración 26 se crearon dos rutas tipo GET para retornar las vistas, tres de tipo Post donde se envían, dos donde los datos recibidos por los formularios son enviados y uno para finalizar las sesiones de usuario.

```
Route::get('/', 'usercontroller@index');

Route::post('/login', 'usercontroller@log');

Route::post('/logout', 'usercontroller@logout');

Route::get('/registro', 'usercontroller@reg');

Route::post('/registro/crear', 'usercontroller@store');
```

Ilustración 26: Rutas definidas de la aplicación web

3.4.3 Vistas

El directorio Resources/Views es el contenedor de nuestras vistas, dentro se crean nuevas vistas o layouts para la aplicación, fueron creadas con la extensión blade.php como se muestra en la ilustración 27. Tenemos tres vistas principales, Welcome para nuestra pantalla principal del sistema y donde se encuentra el formulario de Login, Registro para el formulario de registro y Home para la sesión del usuario Logueado.

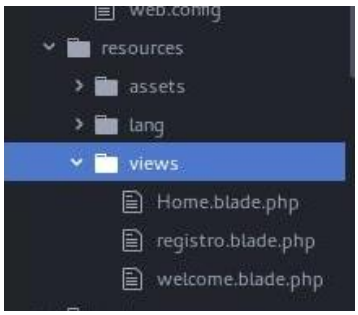


Ilustración 27: Carpeta de vistas de la aplicación web

3.4.3.1 Formulario y Vista Registro

El formulario trabaja de la misma forma que otros, se crea el método POST pero en la acción se agrega un URL para el llamado a la función dentro del controlador que será responsable de la siguiente acción descrito más adelante en la sección del controlador. En caso de que el controlador encuentre un error en la validación de los campos notificará con un mensaje de error dentro la vista de Registro, como se muestra en la ilustración 28.

```
<form method="POST" class="form-signin" action=" {{ url('registro/crear') }} ">
  @if ($errors->any())
  <div class="alert alert-danger">
    <p>Por favor corrija el error</p>
     <ul>
         @foreach ($errors->all() as $error)
           <li>{{ $error }}</li>
         @endforeach
     </ul>
   </div>
   @endif
{!!csrf_field()!!}
```

Ilustración 28: Envío de mensajes de error de la aplicación web

Al final de la notificación de errores se agrego el campo para el *csrf_field*, nuestro token para evitar el exploit con el mismo nombre CRSF. Dentro de las secciones input declaramos los nombres de los valores que vamos recibir en los campos de texto, agregamos validaciones simples HTML como type, required, minlength y maxlength, para tener un filtro básico y simple de los valores de entrada. Finalmente se revisa la pantalla de registro, para verificar que corresponde a los que hemos programado, como se muestra en la ilustración 29.

Ilustración 29: Pantalla de la vista registro de la aplicación

3.4.3.2 Formulario y Vista Login

Para la pantalla de Login se utiliza la misma estructura del Registro, se muestran los errores de validación recibidos por el controlador y se agrega el token *csrf_field*, como se muestra en la ilustración 30.

```
<form method="POST" class="form-signin" action=" {{ url('/login') }} ">
  @if ($errors->any())
  <div class="alert alert-danger">
    <p>Por favor corrija el error</p>
     <ul>
         @foreach ($errors->all() as $error)
           <li>{{ $error }}</li>
         @endforeach
     </ul>
   </div>
   @endif
{!!csrf_field()!!}
```

Ilustración 30: Envío de mensajes de error de la aplicación

De la misma manera campos de input como la validación básica HTML fueron agregadas. Ahora se verifica en la ilustración 31, que nuestra vista de login corresponde a lo descrito en el formulario.

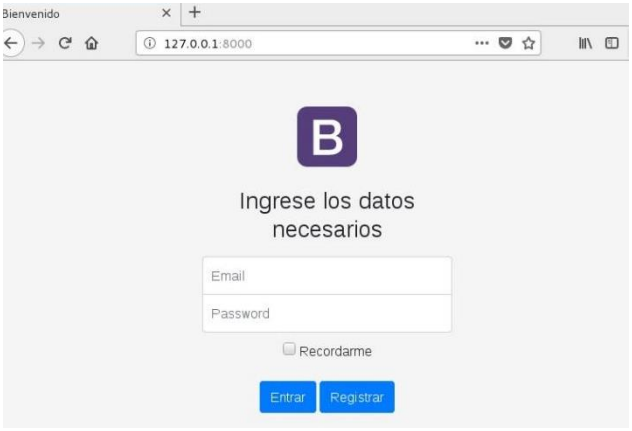


Ilustración 311: Pantalla de la vista de login de la aplicación

3.4.3.3 Formulario y Vista Home

Para finalizar el acceso de un usuario se muestra una pantalla simple donde se inicia una sesión, se agrega el token *csrf_field* y un formulario de cierre de sesión, mostrado en las ilustraciones 32 y 33.

```
<h1 class="panel-title">Bienvenido {{ auth()->user()->name }}</h1>
<form method="post" action="{{ url('/logout') }}">
{!!csrf_field()!!}
<button class="btn btn-primary">Regresar</button>
</form>
```

Ilustración 32: Implementación del campo CSRF_FIELD

Ilustración 33: Vista de validación de usuario

3.4.4 Base de Datos

Para poder trabajar con estos registros y acceso al sistema se creó una ruta a la base de datos del sistema utilizando las variables de entorno de Laravel Framework como se muestra en la ilustración 34.

```
APP_NAME=mipagina
APP_ENV=local
APP_KEY=base64:7RhKGqoeU/FIoO5Mk329H6SQZHJ9Zad3luFvfBYeaDQ=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://mipagina.app

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=clientes
```

Ilustración 34: Definición de la base de datos clientes en el archivo .ENV

La tabla de usuario la creamos utilizando las herramientas de base de datos que nos provee Laravel en su sección de Migraciones, las cuales permiten un manejo sencillo y eficiente de la base de datos que es usado dentro de la aplicación, como se muestra en la ilustración 35, utilizando migraciones se crean las tablas y se añaden las características de estas.

```php
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
```

Ilustración 35: Migración para la creación de la tabla users

La creación y ejecución de la migración corresponde a la base de datos como se muestra en la ilustración 36, un id, name, email, password y tres token para conocer los tiempos de registro o modificación de los usuarios.

```
mysql> DESCRIBE users;
+----------------+------------------+------+-----+---------+----------------+
| Field          | Type             | Null | Key | Default | Extra          |
+----------------+------------------+------+-----+---------+----------------+
| id             | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name           | varchar(50)      | NO   |     | NULL    |                |
| email          | varchar(50)      | NO   | UNI | NULL    |                |
| password       | varchar(255)     | NO   |     | NULL    |                |
| remember_token | varchar(100)     | YES  |     | NULL    |                |
| created_at     | timestamp        | YES  |     | NULL    |                |
| updated_at     | timestamp        | YES  |     | NULL    |                |
+----------------+------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Ilustración 36: Tabla users dentro de la base de datos

3.4.5 Controlador

Como se observa en las rutas, estas dependen de un controlador que es el responsable de las funciones de la aplicación. Por lo que las respuestas a las peticiones Get o Post se realizan a través del uso del controlador.

Las funciones básicas del controlador tipo GET son respuestas de las vistas anteriormente descritas, son peticiones básicas del usuario al servidor, incluyendo la salida de la aplicación por el usuario. Se hace la llamada a la función del middleware Auth para cerrar la sesión y regresar la vista a la ruta inicial.

3.4.6 Validación

Dentro del controlador se tienen funciones más complejas para validar que los accesos al sistema sean correctos, dentro de la función log ilustración 37 podemos observar un request donde se valida que el email contenga una cadena y sea un email de la misma manera que password no pueda ingresar una cadena vacía dentro del sistema, en caso de no pasar la validación respondemos con una cadena notificando el error encontrado. Si la validación resulta correcta se procede a llamar a la sección del middleware responsable de controlar las sesión que verifica la existencia del usuario y la creación del respectivo token de sesión

```
public function log (){
  $logg = request()->validate([
    'email' => ['required','email'],
    'password' => 'required'
  ], [
      'email.required' => 'Introduzca un correo valido',
      'password.required' => 'Introduzca una contrasena valida'
  ]);

  if(Auth::attempt($logg)){
    return view('/Home');
  }
  return back()->withErrors(['email' => 'El usuario no existe']);
}
```

Ilustración 37: Función del método Login en el controlador

Dentro del *required* se validan los valores recibidos dentro del formulario de registro, estos no pueden ser cadenas vacías, en caso del email debe de comprobarse que es un email válido y único, no registrado con anterioridad.

En la ilustración 38 podemos comprobar que al no pasar la validación se envía el mensaje correspondiente de error a la vista.

```
public function store (){

  $data = request()->validate([
    'name' => 'required',
    'email' => ['required','email','unique:users,email'],
    'password' => 'required'
  ], [
      'name.required' => 'Introduzca un nombre valido',
      'email.required' => 'Introduzca un correo valido',
      'email.unique' => 'Este correo ya existe',
      'password.required' => 'Introduzca una contrasena valida'
  ]);

  User::create([
    'name' => $data['name'],
    'email' => $data['email'],
    'password' => bcrypt($data['password'])
  ]);

  return redirect('/');
}
```

Ilustración 38: Función del método Store en el controlador

Finalmente, los datos pasan al modelo crear usuario, este se encarga de agregarlos a la base de datos. Dentro del password se implementa la función Bcrypt para el encriptado de la contraseña en la base de datos, como se muestra en la ilustración 39 todas las contraseñas van protegidas.

```
|  1 | hector  | hector@example.com | $2y$10$V7lsqgOZDCKNNj15naO14ul.uiQyyoczkS/82VHG/
lpRl3GY6P9S | HGuZforEWKbKOSeUSjMDSARnuZaF20herKgWo2Kghm32naa3Pdhl0ituIzoR | NULL
        | NULL                |
|  2 | snow    | snow@example.com   | $2y$10$pGzGWylmCb.bXr0duzsvX.YAbmceSJ.WlOLv/OUfjl
SLn9CemMNHC | NULL                                                       | 2019-03-1
 01:26:39 | 2019-03-17 01:26:39 |
|  3 | Joab    | joab@example.com   | $2y$10$a2SxmTXIdQlVXlQldFwyEOdgZnAuiyuQ8Lc09n0Xp
OQF0JfzFmiK | NULL                                                       | 2019-03-1
 04:57:35 | 2019-03-17 04:57:35 |
|  9 | juan    | juan@example.com   | $2y$10$RW3pOu0JEpMjxh8iE.7cluAaPnVy2CRwmjIkxfAGt
c3YrOJNzG2S | NULL                                                       | 2019-03-1
 16:17:02 | 2019-03-17 16:17:02 |
| 10 | ahri    | ahri@gmail.com     | $2y$10$1c6Nl1SQ8CGK5Pe9rNGMquK8ZtqsBbEwfk3pg5zPo
An3wotPlTRK | NULL                                                       | 2019-03-1
 20:48:02 | 2019-03-17 20:48:02 |
| 11 | luxanna | lux@example.com    | $2y$10$DlpciVhoA1acQxsvbD.bguG//f8EGXzO7GKmzkvm7
J3sBUmAc0Ey | NULL                                                       | 2019-03-1
 01:59:41 | 2019-03-18 01:59:41 |
+----+---------+--------------------+-----------------------------------------------
------------+-------------------------------------------------------------+----------
----------+---------------------+
6 rows in set (0.00 sec)
```

Ilustración 39: Registros realizados y almacenados en la base de datos, contraseñas cifradas

3.4.7 Laravel Framework Log Event y error 404

Una de las funciones que aporta Laravel y Artisan, es la posibilidad de levantar el servidor en una red local, con esta se muestra un log event en tiempo real de lo que sucede dentro del acceso al sistema, ilustración 40.

```
demon@2806-10a6-0008-273a-020c-29ff-fe7e-4986:/var/www/html/mipagina$ php art
isan serve --host 192.168.1.87  --port 8080
Laravel development server started: <http://192.168.1.87:8080>
[Sat Mar  9 15:20:31 2019] 192.168.1.87:55767 [200]: /favicon.ico
[Sat Mar  9 15:20:52 2019] 192.168.1.75:59389 [200]: /favicon.ico
^C
demon@2806-10a6-0008-273a-020c-29ff-fe7e-4986:/var/www/html/mipagina$ php art
isan serve --host 192.168.1.87  --port 8080
Laravel development server started: <http://192.168.1.87:8080>
[Sat Mar  9 15:23:22 2019] 192.168.1.70:41853 [200]: /favicon.ico
[Sat Mar  9 15:25:01 2019] 192.168.1.80:59858 [200]: /favicon.ico
[Sat Mar  9 15:26:49 2019] 192.168.1.72:65454 [200]: /favicon.ico
```

Ilustración 40: Log de accesos a la aplicación web

También permite notificar de errores de acceso a rutas no disponible o no existente, así como de excepciones lanzadas a los usuarios con un error 404, el código se muestra en la ilustración 41.

```
public function render($request, Exception $exception)
{
if($exception instanceof \Symfony\Component\HttpKernel\Exception\Meth
  return abort(404);
}

return parent::render($request, $exception);
}
```

Ilustración 41: Envío de error 404 en excepciones.

Capítulo 4: Desarrollo de un Plan de Auditorías

4.1 Plan de Auditoría

Siguiendo las recomendaciones de OWASP Testing Guide V4 [14] y en base a la escala OWASP Top 10 – 2017 The Ten Most Critical Web Application Security Risk [15], se establece una guía de cómo evaluar las vulnerabilidades en una aplicación web utilizando el Framework Laravel, no es una manual técnico de cómo usar herramientas de penetración, sino una serie de pruebas de evaluación de vulnerabilidades para documentar y evaluar los riesgos dentro de una aplicación web. Como se describe en un plan de auditorías en seguridad los puntos a tratar son:

- Recopilación de información
- Pruebas de gestión de la configuración y la implementación
- Pruebas de gestión de la identidad
- Pruebas de autenticación
- Pruebas de autorización
- Pruebas de la gestión de sesiones
- Pruebas de validación de entrada
- Control de errores
- Criptografía
- Pruebas de lógica empresarial
- Pruebas de lado del cliente

Para continuar con el plan de auditorías es necesario establecer una posición de usuario y agente malicioso, donde se asume un desconocimiento de la estructura interna del sistema, así como del código de la aplicación web. Usando pruebas de Black Box y Grey Box, usando las herramientas de evaluación y penetración de vulnerabilidades: ZAP para un análisis de escáner activo de la aplicación y Burp Suite para trabajar pruebas pasivas de auditoría.

Finalmente se documentan las pruebas realizadas a la aplicación web para crear una evaluación de los riesgos en base a las vulnerabilidades encontradas, basándose en la metodología de evaluación de riesgos OWASP Top 10 – 2017 [15] y OWASP Risk Rating Methodology [16].

4.2 Ataques y Pruebas en el Sitio Web

4.2.1 Recopilación de información

Iniciamos el plan de auditoría haciendo una navegación manual de la aplicación web que se desea evaluar, de forma pasiva con Burp Suite y activa con ZAP, este tipo de exploración dentro de la aplicación que se va a auditar, ayuda a tener un panorama de cómo está respondiendo el sistema, muestra las peticiones que trabaja, los tipos de entrada que recibe y las tecnologías que usa. Burp Suite muestra un árbol de navegación con las carpetas y archivos a los que se han accedido por esta navegación con el proxy, creando un mapa del sitio, como se muestra en la ilustración 42.

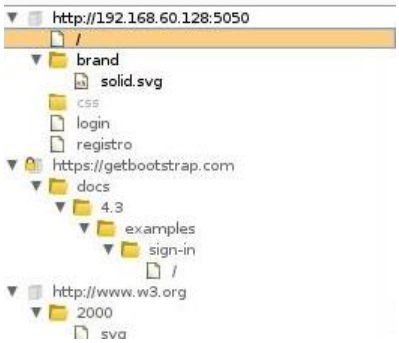


Ilustración 42: Mapeo de la aplicación web con Burp Suite

Mientras que ZAP lo recrea al hacer una exploración activa con el escáner, ilustración 43.

Ilustración 43: Mapeo de la aplicación web con ZAP

Al terminar el scanner activo de ZAP, se identifican distintos métodos GET que corresponden a vistas, así como llamadas de archivos que corresponden al Framework. También se observar dos métodos POST que corresponden al Login y al Registro, junto con las variables que se envían en cada uno de sus respectivos formularios. Esta información será necesaria cuando se comiencen con las pruebas pasivas del Sistema.

4.2.2 Pruebas de gestión de la configuración y la implementación.

Entre las distintas formas para comprobar la configuración y los límites que se pueden trabajar con las peticiones (HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT), es un listado de configuración del sitio como se muestra en la ilustración 44, para ver cuáles son permitidas por este y que tipo de información se puede obtener al trabajarlas.

```
$ nc www.victim.com 80
OPTIONS / HTTP/1.1
Host: www.victim.com

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:00:29 GMT
Connection: close
Allow: GET, HEAD, POST, TRACE, OPTIONS
Content-Length: 0
```

Ilustración 44: Ejemplo de configuración de rutas vulnerable

A diferencia de lo que se espera encontrar, la ilustración 45 se muestra la inexistencia de permisos a otras peticiones, lo que descarta la posibilidad de trabajar con accesos a través rutas, peticiones, así como de una prueba XSRF.

Ilustración 45: Configuración de rutas de la aplicación web

4.2.3 Pruebas de gestión de la identidad

Con el respectivo mapeo del sitio es hora de trabajar con los métodos POST que se encuentra al inicio de sesión y registro de usuarios. En la ilustración 46, se evalúan los distintos escenarios, y agregan campos no esperados a los formularios del sitio para ver su respuesta.

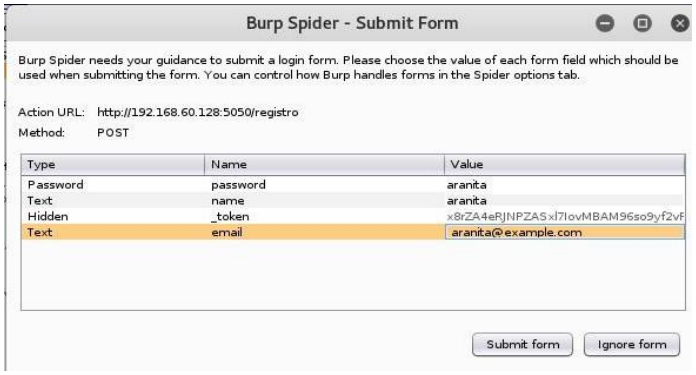


Ilustración 46: Formulario generado por el Spider

Se usa un bot (Spider) para recorrer cada uno de estos formularios, de forma pasiva agregamos distintas entradas y observamos las salidas. Es de resaltar que el sitio no reconoció que había un bot trabajando dentro de su sistema de formularios. Si se comprueba de forma manual la base de datos, se corroborar que se encontró un registro de usuario, así como el login al sistema usando un bot, ver ilustración 47.

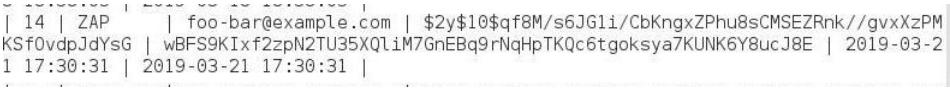


Ilustración 47: Ejemplo de un registro utilizando un bot

Se programó el Spider para trabajar de forma automática, se observó que puede realizar múltiples iteraciones sin que el sistema lo detecte, ilustración 48.

Ilustración 48: Iteraciones realizadas por el Spider

4.2.4 Pruebas de autenticación

Burp Suite trabaja como un proxy y permite interrumpir el envío de mensajes, por lo que los canales de transferencia, GET o POST, pueden ser capturados. En la ilustración 49, se observar que cuando se envía información por formulario no es enviada de forma segura debido a la ausencia de certificados HTTPS, lo que permite ataques conocidos como Man-in-the-middle.

Ilustración 49: Ejemplo de una captura de información por Man in the middle

La transferencia de información del formulario de vista al controlador suele ser la sección más vulnerable del MVC, por que suele ser un punto de mira durante los procesos de auditoría. Es necesario que se pueda certificar los canales de transferencia.

4.2.5 Pruebas de autorización

Si se hacen distintos intentos de entrada probando los diferentes nombres de usuario y contraseña en el sistema web se pueden comprobar en la ilustración 50, que el sitio no cuenta con una excepción de límite de intentos fallidos de entrada, lo que hace posible trabajar algoritmos de

fuerza bruta ya sea con nombres de usuario y contraseñas, lo que hace vulnerable la aplicación web.

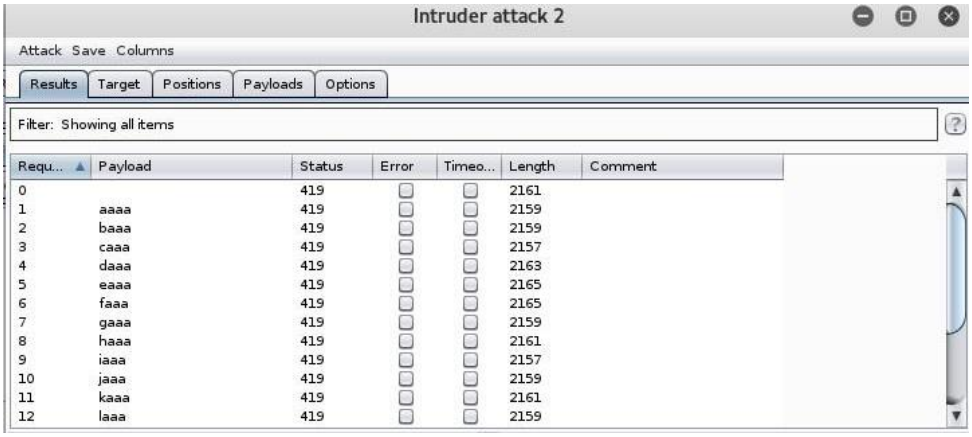


Ilustración 50: Prueba de un generador de contraseñas por fuerza bruta

4.2.6 Pruebas de gestión de sesiones seguras en Laravel

En las pruebas anteriores se muestra que, al iniciar una sesión, ésta queda guardada en un token, es común encontrar elementos de la sesión como el ID y el password en una variable, así como en cifrado hash o en su valor hexadecimal, lo que las hace inseguras cuando son transportadas, al trabajar con tokens de sesión se asegura la protección de los datos, ya que este token que se crea y destruye por lo que la captura de información sesión no está siendo vulnerada. En la ilustración 51 se muestra el token de sesión de Laravel interceptado.

Ilustración 51: Token de sesión de Laravel interceptado

Como se muestra en las ilustraciones 52 y 53 al realizar un login al sistema se dirige a una página exclusiva de acceso donde muestra el nombre del usuario identificado en la sesión.

Internamente se crea un token de sesión donde se guarda la información sensible de acceso.

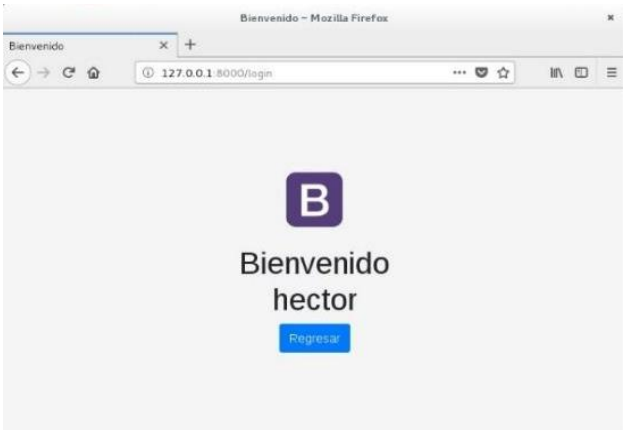


Ilustración 52: Ejemplo de inicio de sesión con Token creado usuario identificado

Al salir el token se destruye por lo que al intentar el acceso directo a la sesión creada enviará un mensaje de error 404. Este tipo de validaciones internas evitan que haya variables o tokens con sesiones abiertas incluso cuando se haya guardado un usuario y contraseña dentro del navegador.

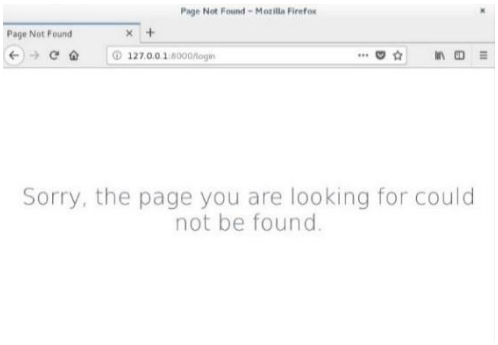


Ilustración 53: Ejemplo de inicio de sesión sin Token

4.2.7 Pruebas de validación de entrada a Bases de Datos con CO2 Plugin

Burp Suite permite utilizar CO2 un plugin para trabajar con SQLMap. Dentro se puede construir diferentes consultas SQL y probarlas dentro de la aplicación que ese quiere auditar.

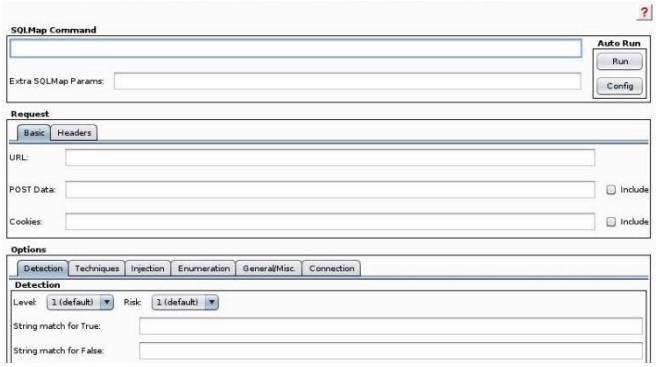


Ilustración 54: Plugin CO2 para SQL Injection

En la ilustración 54 y 55 se muestra que no se permite el uso de peticiones usando métodos no definidos. Por lo que al hacer una petición con código SQL (SQL Injection) por métodos GET o POST resulta fallida.

Ilustración 55: Ejemplo de prueba fallida SQL Map

Para la validación de datos de entrada es necesario realizar iteraciones, por lo que es mejor utilizar herramientas que permiten el escaneo a los métodos de acceso. Se recomienda el uso de Metasploit Framework para probar diferentes escenarios.

4.2.8 Control de errores – Acceso a ruta no autorizadas de la aplicación

En caso de accesos a rutas del sistema no autorizadas o permitidas, así como de rutas que por defecto espera valores en la transferencia de métodos POST, son errores producidos por excepciones ya sean del servidor o la base de datos, la aplicación responde con errores 404. Este lo podemos comprobar con la herramienta Repeater de Burp Suite, ilustración 56, donde podemos

hacer diferentes consultas a la aplicación de forma automática o manual. También se inspeccionaron las rutas dentro del caché compilado del sistema para comprobar los controles de errores.

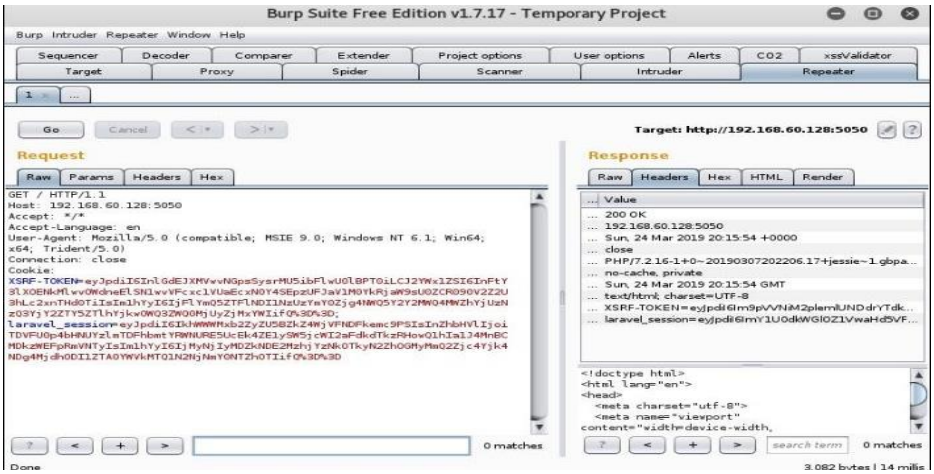


Ilustración 56: Herramienta Repeater de Burp Suite

4.2.9 Criptografía

Como se pudo comprobar en pruebas anteriores es fácil capturar información que no está cifrada o cuando se emplearon métodos de cifrados débiles durante la transferencia, la implementación de otras herramientas para la extracción de tokens con la herramienta decoder de Burp Suite, permite trabajar los tokens que sean capturados durante las pruebas, en la ilustración 57, se muestra un token de sesión, el cual al ser desempaquetado se puede observar su contenido.

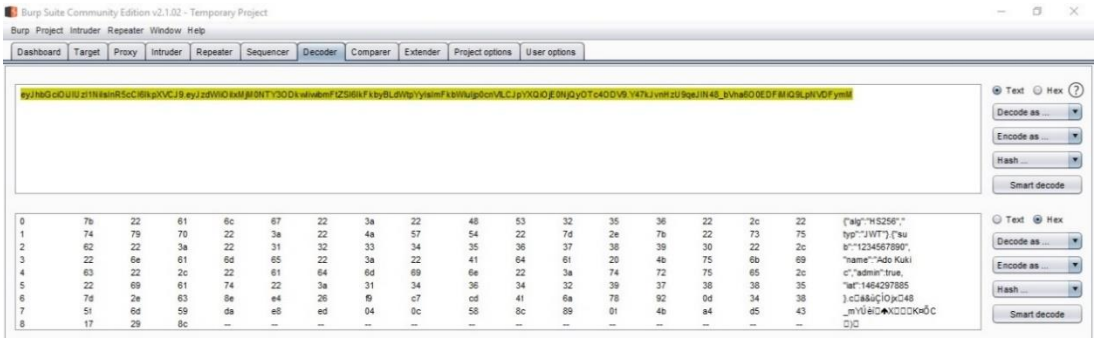


Ilustración 57: Ejemplo de desempaquetado de Token – Burp Suite Decoder

4.2.10 Pruebas de lógica empresarial

Es necesario evaluar las respuestas del sistema a eventos inesperados, agregando varios valores erróneos para probar que las validaciones del sistema funcionen correctamente y que estas no muestran información detallada de los errores, el servidor web, el Framework y la base de datos. dar información técnica de un error a un agente lo puede llevar a saltar los controles de validación, así como de conocer detalles y funciones de las variables que utiliza el sistema

Ilustración 58: Prueba de cadenas vacías y datos aleatorios

Ilustración 59: Prueba de usuarios no válidos

Ilustración 60: Pruebas de cadenas vacías y datos aleatorios

Ilustración 61: Registro con correo repetido

Ilustración 62: Prueba de cadenas vacías y datos aleatorios

En las ilustraciones 58, 59, 60, 61 y 62, se realizaron varias pruebas manuales, en las que se agregaron, cadenas vacías y con valores aleatorios, usuarios ya existentes para los registros, con errores en las contraseñas y nombres de usuarios. La exploración manual nos permite verificar que la aplicación cuenta con funciones de validación basadas en la comparación de los valores que se reciben y donde se comprueban que cumplan con los requisitos del sistema. Se debe destacar que la aplicación web cuenta con la notificación de errores, pero no muestra más información que la necesaria al usuario.

4.2.11 Pruebas del lado del cliente

Hay que tener en cuenta que es posible que se pueda explotar una vulnerabilidad de forma nativa del lado del cliente, ya sea por medio del navegador o por una extensión creada dentro del navegador. En la ilustración 63 se observa un ejemplo de cómo se realiza una prueba de vulnerabilidad del lado del cliente basado en clickjacking, es un ataque donde se usan varias capas transparentes para engañar a un usuario, este realizar click dentro de la aplicación, pero el atacante puede redirigir los click a donde este le parezca mejor. Durante la evaluación usando Burp Suite, utilizando el campo de Burp Clickbandit (es una herramienta para generar ataques de clickjacking) de Burp Suite, dentro del header de la aplicación

Ilustración 63: Explotación por Clickjacking usando Burp Clickbandit

Se observa en la ilustración 64, la herramienta Burp Clickbandit que permite agregar campos ¨falsos¨ o transparentes a la vista de la aplicación, agregado *javascript* mediante la consola del navegador.

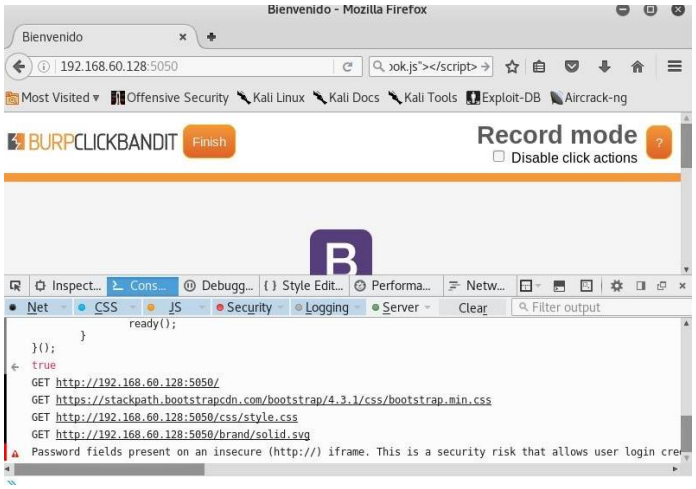


Ilustración 64: Añadiendo campos de clickjacking a la aplicación

Una vez que se agrega la extensión, se logra explotar la vulnerabilidad de la aplicación web, lo permitirá robar el click a un usuario y dirigir al usuario a otros enlaces.

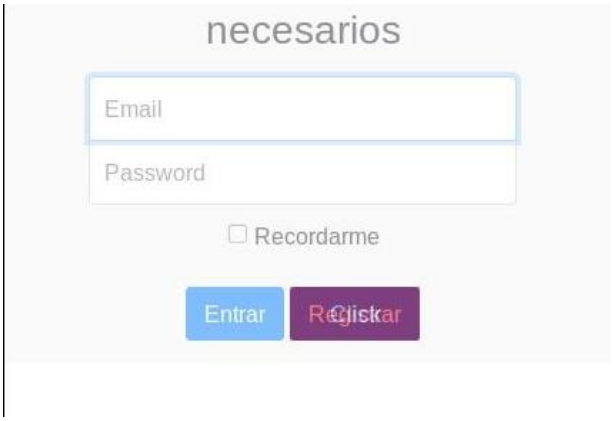


Ilustración 65: Página resultante después de la post explotación con Burp Clickbandit

4.3 Resultados

Las pruebas activas de ZAP cubren un grupo muy general de búsqueda, utiliza distintas pruebas para dar un resultado aproximado de las vulnerabilidades que se podrían encontrar, en la ilustración 66, se observa la respuesta del escaneo activo de ZAP, este nos muestra 6 alertas, 5 de Mínimo riesgo y 1 de riesgo Moderado, es posible que se puedan encontrar más errores realizando pruebas pasivas, como las realizadas con Burp Suite. Tras finalizar estas pruebas pasivas de Burp y activas de ZAP, tenemos un panorama general de los riesgos encontrados en la aplicación web y se puede empezar a construir un reporte usando como referencia las recomendaciones a OWASP Risk Rating Methodology y OWASP Top 10 – 2017.

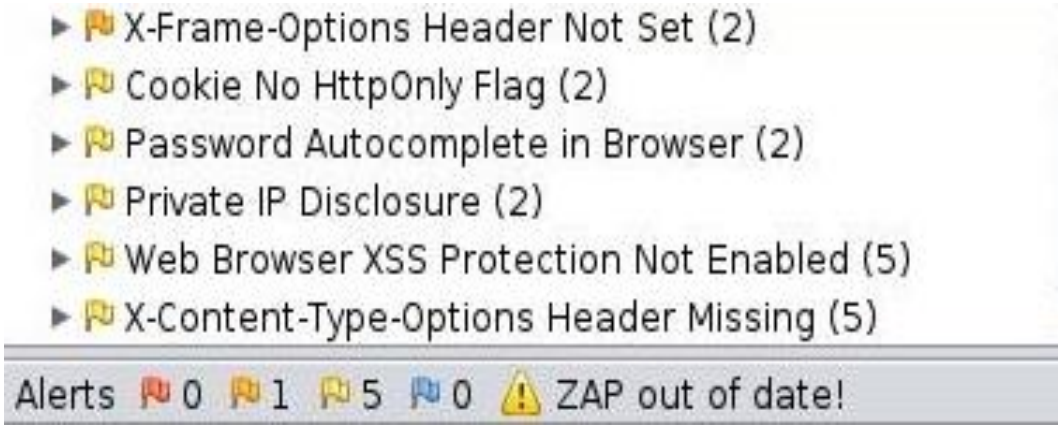


Ilustración 66: Mensaje de alertas enviadas por el escáner de ZAP

4.3.1 Resultado de ZAP

Para proceder con nuestra evaluación de riesgos, se definen tres factores que involucran directamente a una vulnerabilidad: el vector de ataque, la vulnerabilidad en seguridad del sistema y el impacto técnico. Usando la metodología de evaluación de OWASP se crea una tabla de contenido basado en el diagrama de evaluación, ilustración 67 y le asignamos un valor entre el 1 y 3, y un código de color que puede ser amarillo, naranja o rojo para describir dentro de la tabla.

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli-cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

Ilustración 67: Evaluación de riesgos [14].

El vector de ataque está directamente relacionado con la vulnerabilidad, ya que nos da un panorama de cómo se podría explotar la vulnerabilidad encontrada, que implica la existencia de esta y por lo tanto es necesario describir una definición básica de lo que consiste la vulnerabilidad. El número 1 y amarillo implica que la vulnerabilidad es difícil de explotar, 2 y anaranjado da como resultado una vulnerabilidad complicada pero posible de explotar y finamente 3 con rojo es una vulnerabilidad sencilla de explotar.

La vulnerabilidad en seguridad del sistema está relacionada con dos características, la prevalecía de las vulnerabilidades, qué tan común o raro es encontrar esta vulnerabilidad, 3 con rojo para vulnerabilidades generalizadas, 2 con anaranjado para comunes y 1 con amarillo para prevalencias raras. La prevalencia de una vulnerabilidad es una característica particular, implica que a pesar de ser posible encontrar la vulnerabilidad si es rara su prevalencia dentro de un sistema podría ser descartada de las evaluaciones y planes de auditoría. El qué tan detectable es una vulnerabilidad es una característica que implica la dificultad o facilidad de encontrar esta vulnerabilidad, la existencia de herramientas para la detección de esta vulnerabilidad, la dificultad de realizar las pruebas, el tiempo y recursos son tomados en consideración para asignar una calificación, 3 con rojo para vulnerabilidades fáciles de detectar, 2 con anaranjado para moderadas y 1 con amarillo para difíciles.

El impacto técnico que tiene la vulnerabilidad hace referencia a los daños que se podrían llegar a alcanzar tras la explotación, por lo que evaluar en dónde se encuentra la vulnerabilidad dentro del

sistema, hasta donde se puede escalar y qué sectores se verían afectados tienen que ser tomados en consideración. Se mide con 3 y rojo para impactos técnicos severos, 2 con anaranjado para moderados y 1 con amarillo para mínimos. Los reportes de OWASP miden también el impacto de negocios, el cual consiste en evaluar el daño que se podría llegar a alcanzar a un nivel muy específico dentro del sistema, si este es usado por una empresa, organización gubernamental o bancaria, así como el valor de la información comprometida, recursos, etc. Para esta evaluación no se toma en cuenta el impacto de negocios ya que las vulnerabilidades del sistema evaluado no representan un daño real, pero debe de tratarse en un plan de auditorías, por lo que ahora se muestra la evaluación de las vulnerabilidades encontradas durante en escaneo activo con ZAP.

Tabla 1: Vulnerabilidad encontrada con ZAP; X-Frame-Options Header Not Set

X-Frame-Options Header Not Set			
Vectores de Ataque	**Vulnerabilidades en Seguridad**		**Impacto**
Explotable: 3	**Prevalencia: 2**	**Detectable: 2**	**Riego Técnico: 2**
El error consiste en la ausencia de una cabecera X-Frame-Options, lo que permite la explotación mediante Clickjaking	Es una vulnerabilidad del lado del cliente, por lo que es fácil solucionarlo	Es una vulnerabilidad fácil de encontrar, con el simple análisis de los headers de la página se puede encontrar esta vulnerabilidad	Puede permitir el robo de información como password, permite la redirección a páginas sospechosas.

Tabla 2: Vulnerabilidad encontrada con ZAP; Cookie No HttpOnly Flag

Cookie No HttpOnly Flag			
Vectores de Ataque	**Vulnerabilidades en Seguridad**		**Impacto**
Explotable: 1	**Prevalencia: 1**	**Detectable: 2**	**Riego Técnico: 2**
Es un error conocido por la ausencia del HttpOnly en los encabezado de respuesta, la existencia de esta bandera mitiga el riesgo de XSS del lado del cliente	Es un vulnerabilidad conocida en los navegadores.	Es una vulnerabilidad conocida pero no ampliamente documentada ya que los nuevos navegadores suelen incluir una protección para evitar esto	La existencia de esta vulnerabilidad permite la explotación XSS enfocados a tokens de sesión

Tabla 3: Vulnerabilidad encontrada con ZAP; Password Autocomplete in Browser

Password Autocomplete in Browser			
Vectores de Ataque	**Vulnerabilidades en Seguridad**		**Impacto**
Explotable: 1	**Prevalencia: 3**	**Detectable: 1**	**Riego Técnico: 2**
La existencia de nombres de usuario y contraseñas guardados dentro del navegador, son un objetivo para la captura de información sensible	Es una vulnerabilidad basada en las opciones de guardado de sesión en el navegador	Se requiere de una inspección del lado del cliente dentro de su navegador, por lo que se convierte en una vulnerabilidad nacida de la post explotación	El robo de contraseñas y nombres de usuario, permiten al agente tener accesos a diferentes cuentas que hayan sido guardadas en el navegador

Tabla 4: Vulnerabilidad encontrada con ZAP; Private IP Disclosure

Private IP Disclosure			
Vectores de Ataque	**Vulnerabilidades en Seguridad**		**Impacto**
Explotable: 2	**Prevalencia: 1**	**Detectable: 3**	**Riego Técnico: 2**
Existen entradas reservadas para su uso dentro de una red privada y no se deben de enrutar de manera pública ya que permiten el sniffing, spoffing y la denegación de servicios	Esta vulnerabilidad se encuentra en la falta de una configuración específica del servidor para trabajar en una red pública	Se puede detectar la vulnerabilidad con una exploración de los protocolos básicos IP con los que se configuró el servidor.	Robo de información, denegación de servicios, inhabilitación del servidor.

Tabla 5: Vulnerabilidad encontrada con ZAP; Web Browser XSS Protection Not Enabled

Web Browser XSS Protection Not Enabled			
Vectores de Ataque	**Vulnerabilidades en Seguridad**		**Impacto**
Explotable: 2	**Prevalencia: 1**	**Detectable: 1**	**Riego Técnico: 2**
Es una vulnerabilidad nacida de la creación de herramientas dentro del navegador para protegernos de ataques XSS	Dentro de los navegadores de nueva generación tenemos la posibilidad de habilitar y deshabilitar herramientas para la protección XSS	Es necesario hacer pruebas a las herramientas del navegador incluido la protección anti-XSS para encontrar una vulnerabilidad que permita la explotación de la herramienta	La explotación por XSS va desde el robo de datos hasta la inyección de payloads

Tabla 6: Vulnerabilidad encontrada con ZAP; X-Content-Type-Options Header Missing

X-Content-Type-Options Header Missing			
Vectores de Ataque	**Vulnerabilidades en Seguridad**		**Impacto**
Explotable: 3	**Prevalencia: 2**	**Detectable: 1**	**Riego Técnico: 2**
Esta vulnerabilidad está presente en la ausencia de X-Content-Type-Options lo que permite la explotación por clickjaking,	La ausencia de esta protección se encuentran en diferentes aplicaciones debido al desconocimiento.	Existen distintas herramientas que no dan la información suficiente para encontrar esta vulnerabilidad, la exploración manual del código con el navegador es una.	La explotación por clickjaking permite la redirección a páginas infectadas, robo de información a través de la falsificación del sitio.

Las alertas lanzadas por la exploración activa de ZAP, ilustración 66, muestran una carencia en la configuración del servidor y de las herramientas del navegador. Estas pueden ser resueltas con actualizaciones y ajustes en la configuración por defecto del servidor. Pero en específico la ausencia de tres cabeceras de seguridad HTTP (X-Content-Type-Options, X-Frame-Options y X-XSS-Protection), estas cabeceras pueden ser añadidas dentro de las funciones de Laravel Framework de la manera que se muestra en la siguiente ilustración 68 para solucionarlas.

```
$response = $next($request);
$response->headers->set('Referrer-Policy', 'no-referrer-when-downgrade');
$response->headers->set('X-Content-Type-Options', 'nosniff');
$response->headers->set('X-XSS-Protection', '1; mode=block');
$response->headers->set('X-Frame-Options', 'DENY');
$response->headers->set('Strict-Transport-Security', 'max-age=31536000; includeSubDomains');
$response->headers->set('Content-Security-Policy', "style-src 'self'"); // Clearly, you will
```

Ilustración 68: Ejemplo de cabeceras de seguridad Laravel

4.3.2 Resultados de auditoría con Burp Suite

Durante las pruebas pasivas con Burp Suite encontramos varias fallas de diseño en la aplicación web que se listan a continuación.

- Ausencia de una validación de usuarios en login y registro

- Ausencia de un límite de intentos en el inicio de sesión
- Ausencia de certificados HTTPS
- Ausencia de las cabeceras de seguridad
- Ausencia de una configuración adecuada del servidor

Finalmente se asigna un valor de riesgo mínimo a la aplicación web, la mayor parte de las vulnerabilidades encontradas corresponden al diseño y la falta de algunas herramientas de seguridad dentro de la aplicación web, como por ejemplo validaciones tipo *captcha* para distinguir usuarios físicos de bots, certificados en capas de transporte, limites en tiempos de sesión, limites en el intento de acceso de sesión, implementación de recomendaciones para la creación de contraseñas, seguras pero pueden ser cubiertas implementado las recomendaciones propuestas en el siguiente Plan de Auditorias.

Capítulo 5: Conclusiones

El proceso de investigación que se trabajó durante este proyecto de tesis da un panorama muy particular de la seguridad en las aplicaciones web, en base a las pruebas realizadas obtenemos como resultado que:

1) Es recomendable utilizar un Framework del lado del servidor (Back-End) como Laravel Framework, para hacer uso de las herramientas que incluye, para el manejo de sesiones con tokens, cifrado de contraseñas, manejo de base de datos con migraciones, rutas con cierres y middleware, implementación de un csrf_token, validación en las consultas a las bases de datos con eloquent ORM, log de eventos, evaluación de funciones y métodos de la aplicación web con tests, así como implementando mejoras a los procesos de desarrollo de una aplicación web con el modelo RMVC.
2) Consultar la documentación y los cambios que tienen las diferentes versiones de un Framework mantendrá actualizados de las nuevos los paradigmas y la tecnología para el desarrollo, con el objetivo de poder blindar una aplicación web.
3) Después de documentar, implementar y evaluar las herramientas que utiliza Laravel Framework se comprueba que es un Framework que cubre los requisitos básicos de seguridad, las herramientas que incluye junto con su metodología de desarrollo RMVC funciona correctamente, la funciones que se pueden usar del Framework se encuentran bien documentadas y son fáciles de usar, por lo que se recomienda trabajar con Laravel Framework en proyectos profesionales de desarrollo web.

La seguridad es un campo muy amplio y complejo de la computación, específicamente la seguridad web, que se ha convertido en una necesidad para la protección de la información. La creación de nuevos métodos de evaluación y herramientas de penetración crece todos los días y evoluciona

constantemente, tener un conocimiento básico de lo que implica la seguridad nos ayudará a mejorar el proceso de diseño, desarrollo e implementación de aplicaciones web. El uso de herramientas sencillas para la evaluación de vulnerabilidades puede aportar un valor extra a los procesos de desarrollo y mantenimiento de un sistema, dando una perspectiva nueva de cómo se pueden enfocar los esfuerzos y recursos en el diseño, desarrollo e implementación de una aplicación web. Se recomienda el uso de:

1) OWASP Zed Attack Project (ZAP) una herramienta gratuita y documentada. Perfecta para personas que se inician en el campo de la seguridad informática específicamente en la auditoría de aplicaciones web. Su interfaz es sencilla y multiplataforma, las pruebas que se realizaron se hicieron desde Kali Linux y Windows 10.
2) Burp Suite una herramienta profesional de evaluación de vulnerabilidades y pen testing, para aplicaciones web multiplataforma, que nos permite trabajar las pruebas manuales de forma muy sencilla y con la posibilidad de explotar las vulnerabilidades encontradas, así como hacer uso de distintas herramientas de desempaquetado de tokens, ataques de fuerza bruta, mapeos del sitio con bots, etc. Las pruebas que se realizaron se hicieron desde Kali Linux y Windows 10.

La implementación de un plan de auditorías tiene como propósito ayudar a los desarrolladores y organizaciones a identificar los pasos que necesitan seguir para construir su propio programa de evaluación de vulnerabilidades, una serie de requisitos que debe cubrir un plan de auditorías especialmente diseñado para el sistema que están desarrollando. Es por esto que se recomienda la implementación de un plan de auditorías desde el inicio de la creación de un proyecto y se propone seguirlo, aunque debe tener en consideración que no existe una receta única para todos los sistemas.

5.1 Plan de Auditoría en Seguridad Preventivo

Como se observar la ilustración 69, este plan de auditorías preventivo cuenta con 5 puntos y tiene como objetivo su implementación durante los inicios del proceso de desarrollo de una aplicación web.

Ilustración 69: Plan de auditoría preventivo

El primer punto de nuestro plan de auditorías preventivo propone implementar una metodología de desarrollo de software basado en el ciclo de vida del desarrollo de software (Software Development Life Cycle, SDLC, ilustración 70). La implementación un SDLC donde se incluyen filtros de seguridad en primeras fases de la metodología reducirá las ventanas de vulnerabilidad que se podrían encontrar durante las etapas finales del proceso del SDLC, ya que durante las etapas

de análisis y diseño se pueden encontrar y evitar la mayor cantidad de los posibles errores que se descubren durante una evaluación de vulnerabilidades usando herramientas basadas en pen testing.

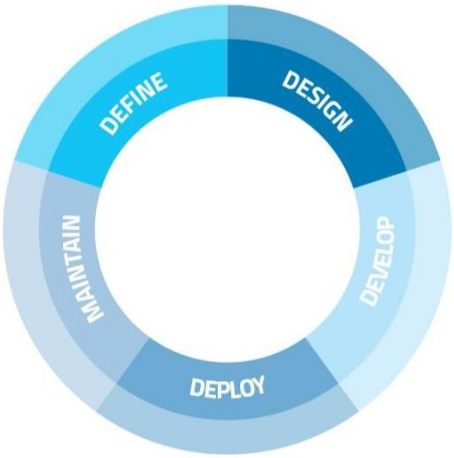


Ilustración 70: SDLC [14].

Para esto es necesario iniciar el proceso de documentación del sistema durante la planificación o análisis. La documentación debe de incluir secciones donde se describen todas las herramientas usadas, como programas, Frameworks, servidores web, lenguajes de programación, sistemas operativos, con notas de las versiones de software, herramientas de hardware, etc.

Durante el segundo punto el proceso de diseño de la aplicación web, se debe de establecer una serie de requisitos en seguridad basados en las tres principales características que deben poseer los datos de una aplicación web:

Confidencialidad: Los datos del usuario se deben de proteger de robo, alteración o eliminación, para esto es necesario establecer validaciones en el control de usuarios y sesiones, estas validaciones van desde la implementación de sesiones cerradas, donde los datos del usuario esta empaquetados en algoritmos de cifrado fuertes, sesiones donde solo se guardan los datos necesarios y mínimos del usuario, evitar tener números de cuentas o tarjetas bancarias, contraseñas de cuentas con información sensible, etc. La autenticación del usuario debe de cumplir con

validaciones que verifiquen la identidad del usuario y se le notifique en caso de cambios y eliminaciones de datos del usuario, en caso de tener niveles de usuario dentro del sistema, trabajar con credenciales adicionales a las del inicio de sesión del usuario.

Integridad: Los datos del usuario siempre deben de enviarse por canales certificados y cifrados, en caso de que la transmisión sufra una interrupción o los datos no sufran una alteración durante su transmisión estos deben de ser eliminados e iniciar el proceso de transmisión nuevamente, para esto es necesario usar protocolos, certificados y canales de transmisión seguros y de ser posible monitoreados, con herramientas como los registros de eventos.

Disponibilidad: El usuario debe de tener el control de cuando consulta, modifica y elimina sus datos de la aplicación web, por lo que es necesario que la aplicación notifique al usuario todas las alteraciones del sistema que sus acciones provoquen, mensajes donde se muestre solo la información necesaria al usuario sin dar demasiada información del cómo funciona el sistema interno.

Todos estos requisitos deben ser tomados en cuenta durante la construcción de los diagramas UML y casos de uso del sistema. De manera que se podrán identificar los parámetros de seguridad que nuestra aplicación está cubriendo y en base a qué estándar se va a evaluar. Diagramas del funcionamiento interno de la aplicación, métodos y funciones, que herramientas del Framework fueron usadas y donde fueron implementadas. Todo esto con el fin de que durante las pruebas de seguridad se puedan cubrir todas las posibilidades, así como también partes más esenciales y específicas de la aplicación web.

El tercer punto cubre el proceso de desarrollo de la aplicación web, donde se propone la implementación de un Framework del lado del servidor (Back-End) como Laravel Framework,

pero los equipos de desarrollo pueden proponer y utilizar algún otro Framework del lado del servidor, mientras este cuenta con herramientas que permitan las pruebas de lógica interna (Test) del sistema, los Test de Laravel permiten la evaluación de métodos y funciones de la aplicación web, en los Test se espera encontrar que la función responder de la manera que se esperar, así como de validar cada una de las entradas que reciben.

Las pruebas manuales de caja de la aplicación web en especial el black box testing basado en el método de fuzzing, es uno de los filtros de seguridad que el desarrollador puede implementar de manera rápida y continua dentro de la aplicación web, en búsqueda de comportamientos inesperados o controles de error no implementados durante los casos de uso.

En la implementación se propone integrar el punto 4, la aplicación de pruebas de penetración (pen testing) activas y pasivas con ZAP y Burp Suite, con el objetivo de encontrar los errores, fallas de diseño y programación de la aplicación web. Las pruebas que se realizaran deben de cubrir 11 puntos esenciales de la seguridad web, los puntos a tratar son:

- Recopilación de información
- Pruebas de gestión de la configuración y la implementación
- Pruebas de gestión de la identidad
- Pruebas de autenticación
- Pruebas de autorización
- Pruebas de la gestión de sesiones
- Pruebas de validación de entrada
- Control de errores
- Criptografía
- Pruebas de lógica empresarial
- Pruebas de lado del cliente

La cantidad de pruebas no está establecida debido a la diferencia entre sistemas y las herramientas que usan, pero se deben de trabajar tantas como sean necesarias como para cubrir los requisitos de seguridad establecidos al principio del diseño de la aplicación. También es necesario verificar la configuración del sistema, estas pruebas no están relacionadas directamente con el funcionamiento de la aplicación, pero sí indirectamente con esta, por lo que debemos de buscar que el sistema no trabaje con la configuración por defecto, cuente con los certificados necesarios, firewall, configuración adecuada de puertos y de direcciones IP.

Se debe de documentar cada una de estas pruebas para mantener un registro de las evaluaciones que se realizaron al sistema, de esta manera se podrán verificar los ajustes y correcciones al sistema, también se podrán comparar resultado de futuras pruebas de penetración en busca de patrones de fallos o nuevas vulnerabilidades.

Finalmente, en el punto 5, se realiza un respaldo y mantenimiento cada determinado tiempo a la aplicación web, el mantenimiento va desde una actualización de software, instalación de un parche o el cambio a nuevas versiones, se debe de trabajar con pruebas de performance, en busca de que no haya nuevos errores, asegurarse que las validaciones implementadas trabajan correctamente y las herramientas de seguridad no presentan fallos. Siguiendo este plan de auditoría se pueden establecer factores de riesgo a la aplicación web, así como de mantener un registro para la elaboración de nuevas pruebas de penetración.

Se debe comprender que la seguridad es un proceso no un producto y no existe una única solución, por lo que se deben usar tantas herramientas sea posible y agregar tantos filtros de seguridad sean necesarios, para poder obtener el mejor resultado posible, donde las ventanas de vulnerabilidad sean mínimas o nulas

5.2 Trabajo Futuro

Se propone la implementación de un servidor donde se pueda trabajar solo Laravel Framework en diferentes sistemas operativos, Windows Server, Mac OS, Linux/Unix y ambientes de trabajo como Laravel Homestead y Laravel Valet con el fin de comparar los ambientes de trabajo y encontrar las mejores configuración de seguridad para una aplicación web desarrollada en Laravel, realizar el plan de auditoría con el fin evaluar la configuración y el cómo se desempeña el Framework estos entre sistemas operativos y ambientes de trabajo.

La implementación de pruebas de penetración en otros Framework Back-End como Ruby on Rails, Django, Microsoft .NET, etc., para crear una comparación entre los Framework basados en diferentes lenguajes de programación. Resultaría interesante implementar y documentar otras herramientas de auditoría y pruebas de penetración como MetaSploit Framework para obtener nuevos resultados en la prueba de penetración y compararlo con los obtenidos con ZAP y Burp Suite, usar diferentes Sistemas Operativos basados en pruebas de penetración y documentarlo.

Implementar pruebas a una aplicación ya liberada y en un proceso de mantenimiento, podría documentarse para mostrar un cambio de una aplicación insegura a su rediseño en una que cumpla un estándar de seguridad propuesto.

Finalmente, este trabajo de tesis se busca incentivar y proponer a otras personas a iniciar nuevos procesos de desarrollo utilizando Framework, implementar metodologías de desarrollo donde se trabajen requisitos de seguridad y documentar las distintas herramientas basadas en pruebas de penetración.

Anexo: Manual de instalación de Laravel Framework en Ubuntu 18 y Debian 8.

Paso 1: Instalar Composer

Laravel Framework necesita un manejador e instalador de dependencias PHP, por lo que la documentación nos pide instalar composer, dirijase al navegador web y agregue la dirección (https://getcomposer.org/download/).

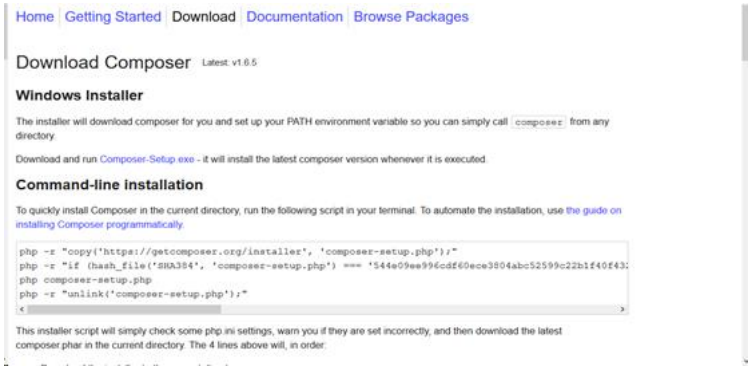


Imagen 1: Página web de Composer

Ahora observe en la imagen 1, que Composer nos da una serie de comandos para ejecutar en terminal y proceder con la instalación.

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash_file('SHA384', 'composer-setup.php') === '544e09ee996cdf60ece3804abc52599c22b1f40f4323403c44d44fdfdd586475ca9813a858088ffbc1f233e9b180f061') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"

php composer-setup.php

Siga los pasos y proceda con la instalación desde la terminal, imagen 2 y 3:

Imagen 2: Instalación de Composer en Debian 8

Imagen 3: Instalación de Composer en Ubuntu 18

Para ejecutar Composer ubíquese dentro de la carpeta donde instalo Composer y desde terminal ejecutamos el siguiente comando*: php composer.phar*, imagen 4 y 5.

Imagen 4: Ejecución de Composer en Ubuntu 18

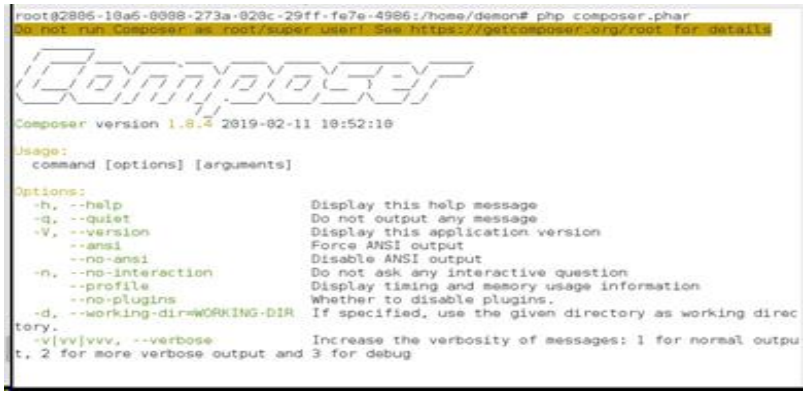


Imagen 5: Ejecución de Composer en Debian 8

Paso 2: Instalar Laravel Framework.

Con Composer instalado, proceda a ejecutar el siguiente comando: *php composer.phar global require ¨laravel/installer¨,* imagen 6 y 7.

Imagen 6: Instalación de Laravel Framework en Ubuntu 18

Imagen 7: Instalación de Laravel Framework en Debian 8

Paso 3: Crear un proyecto

Después de unos minutos tendrá Laravel instalado y podrá proceder a crear el primer proyecto, diríjase a la carpeta siguiente (/var/www/html), abra la carpeta con la terminal y proceda a ejecutar el siguiente comando: *php composer.phar create-project –prefer-dist laravel/laravel ver/www/html/nombredelproyecto*, imagen 8 y 9, para crear un nuevo proyecto, así como para descargar en instalar todos los paquetes necesarios.

Imagen 8: Creación de un proyecto de Laravel en Ubuntu 18

Imagen 9: Creación de un proyecto de Laravel en Debian 8

Finalmente se puede empezar a trabajar en Laravel. Para iniciar nuestro servidor en Laravel proceda a ejecutar el comando: *php artisan serve*, desde la terminal, imagen 10 y 11.

Imagen 10: Ejecutando el servidor de Laravel en Ubuntu 18

```
demon@2806:10a6:0008:273a:020c:29ff:fe7e:4986:~/var/www/html/prueba1$ php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
```

Imagen 11: Ejecutando el servidor de Laravel en Debian 8

Este comando nos devuelve una dirección donde podremos verificar nuestro servidor y la aplicación web ejecutándose, imagen 12 y 13.

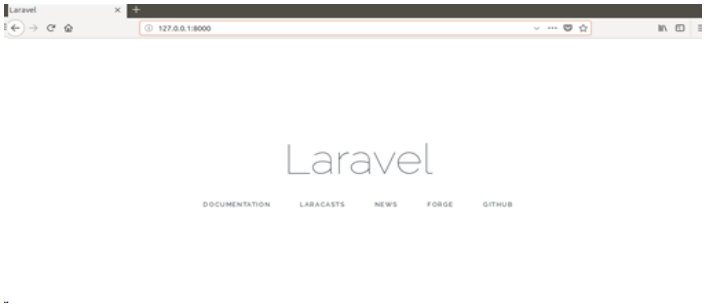


Imagen 12: Aplicación web - Laravel Framework en Ubuntu 18

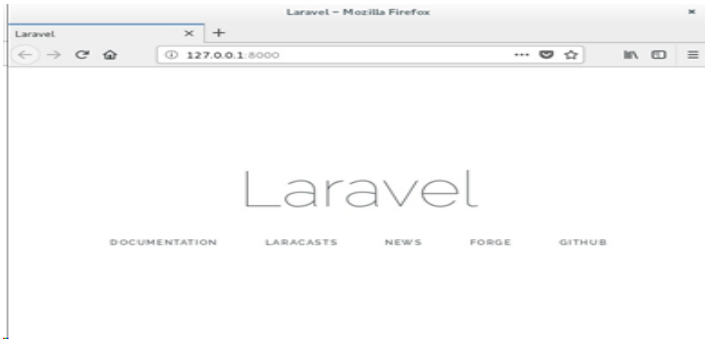


Imagen 13: Aplicación web - Laravel Framework en Debian 8

Anexo: Manual de Instalación de Kali Linux en VMWare.

Kali Linux es un sistema operativo desarrollado como una herramienta avanzada para realizar pruebas de penetración y auditoría. Se puede obtener una copia del sistema de forma gratuita solamente con acceder a sus página web, https://www.kali.org/, en la pestaña de Downloads podemos encontrar diversas versiones de su sistema, elegimos la que se adecue más a nuestras necesidades y procedemos con la descarga, imagen 14.

Imagen 14: Página web de Kali Linux

Kali Linux es un sistema operativo con base Linux, pero que fue creado a partir de una distribución de Debian por lo que su durante su instalación es necesario configurarlo como tal.

Paso 1: Configurar VMWare para trabajar con Kali Linux

Procedemos a crear una nueva máquina virtual. y elegimos una configuración recomendada, imagen 15.

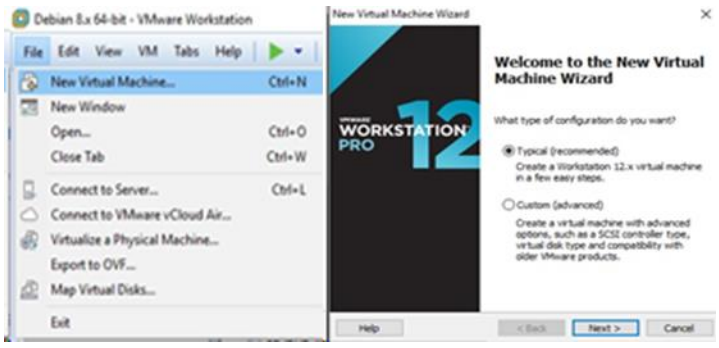


Imagen 15: Creación de una máquina virtual en VMWare

Al seleccionar la imagen descargada del Kali Linux y agregarlo para la instalación, se notificará que el sistema operativo no ha sido reconocido, imagen 16, pero procedemos configurar el SO.

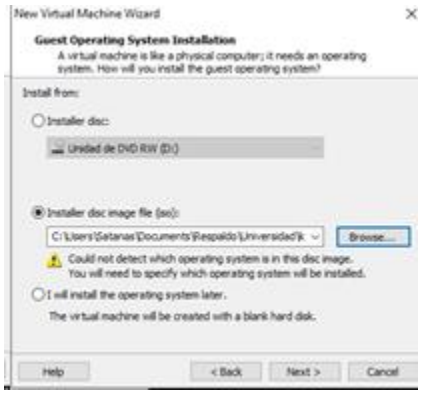


Imagen 16: Agregar imagen de Kali Linux

Por lo que presionamos siguiente y agregamos en la configuración del sistema operativos Linux y seleccionamos la versión Debian 7 con la correspondiente arquitectura de nuestro procesador, imagen 17, presionamos continuar, agregado nombre de la máquina virtual que creamos y siguiente

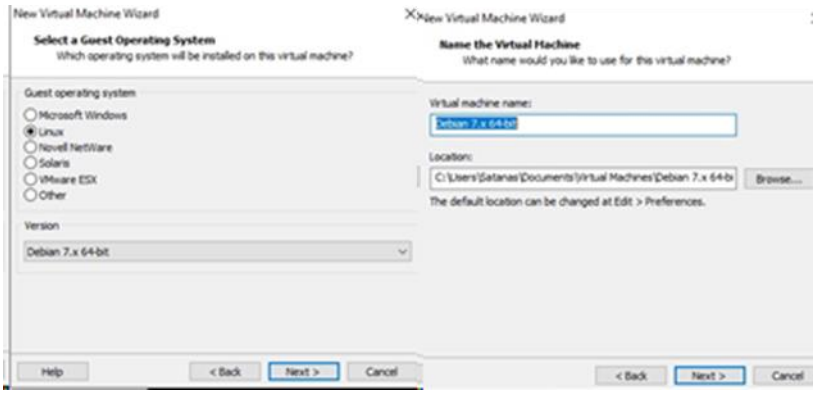


Imagen 17: Configuración del SO dentro de VMWare

Para finalizar con la configuración de la máquina virtual, es necesario dar las especificaciones del hardware que le se asignará para su correcto funcionamiento. 25 GBits para el almacenamiento general del sistema operativo, imagen 18.

Imagen 18: Asignación de VHD en VMWare

Así como también la asignación de 2 GBits para la memoria Ram, agregaremos un procesador de los disponibles con la capacidad de agregar y segundo núcleo por procesador, imagen 18, y terminamos la configuración.

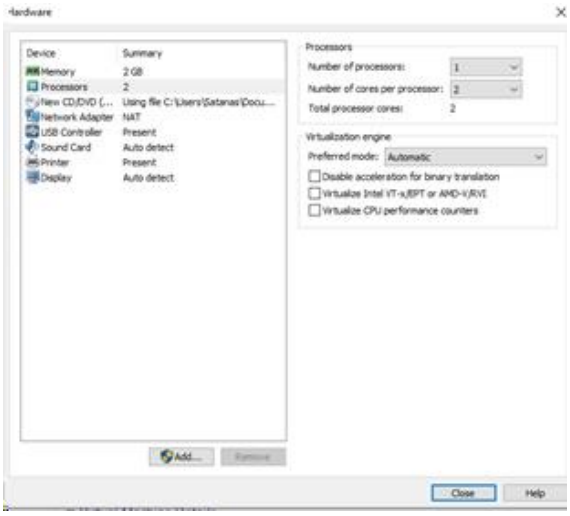


Imagen 19: Configuración de memoria RAM y asignación del número de procesadores

Paso 2: Instalación de Kali Linux

Inicie la máquina virtual, la cual mostrará la siguiente pantalla, imagen 20, seleccionamos la opción Graphical Install.

Imagen 20: Menú de inicio de Kali Linux

Las siguientes pantallas nos pedirá seleccionar el idioma, la región, así como la configuración del teclado, esta selección de opciones depende totalmente de nuestra elección.

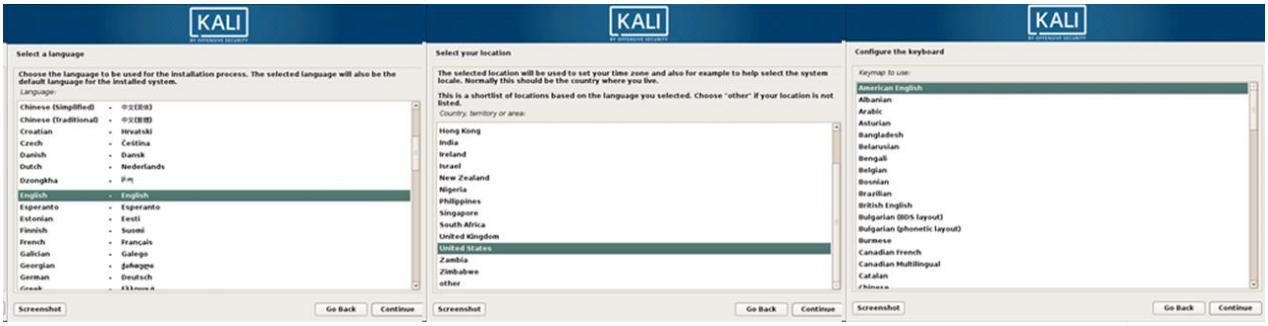


Imagen 21: Configuración de idioma, región e idioma del teclado

Después de elegir la configuración básica, procedemos con la configuración de la contraseña para la cuenta root del sistema, imagen 22. Tenga en cuenta que la contraseña del root es diferente a la de un usuario normal de otros sistemas operativos Linux, ya que este es un super usuario constante en el sistema, por lo que es esencial tener en cuenta que no se podrá entrar, cambiar o recuperar un sistema de Kali Linux al perder u olvidar la contraseña, por el cifrado que utilizan los sistemas Linux para las cuentas de root. Se recomienda usar contraseñas fuertes y robustas ya que estamos trabajando con un sistema de auditoría y penetración así que la seguridad debe de ser una prioridad.

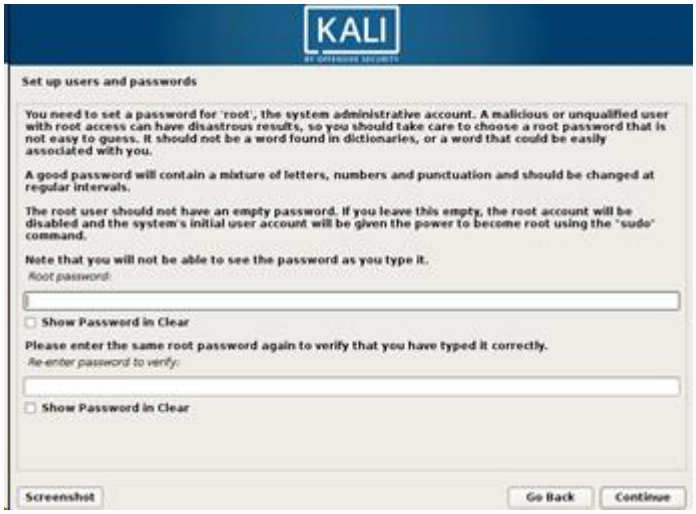


Imagen 22: Configuración de la contraseña del Root

La siguiente configuración corresponde a donde y como almacenaremos el sistema, imagen 23 y 24, se recomienda que para usuarios expertos crear particiones para cada una de las carpetas del sistema, en el siguiente caso para la instalación básica seguiremos la recomendación del sistema.

Imagen 23: Configuración del disco duro 1

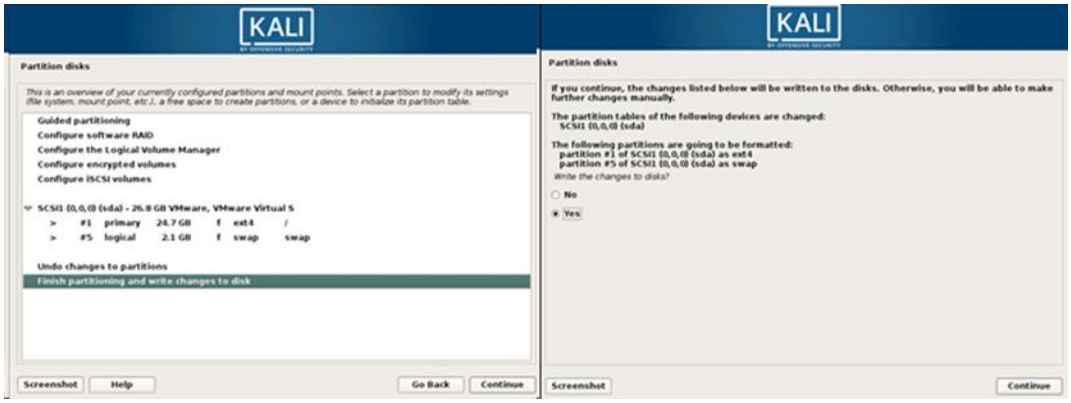


Imagen 24: Configuración del disco duro 2

Cuando finalice de configurar los disco y particiones, la instalación comenzará y tras esperar unos momentos el sistema notificará que no ha encontrado otro sistema operativo instalado, así como una partición de arranque del sistema, es necesario agregar un boot para Kali así se seleccionara GRUB por default, imagen 25.

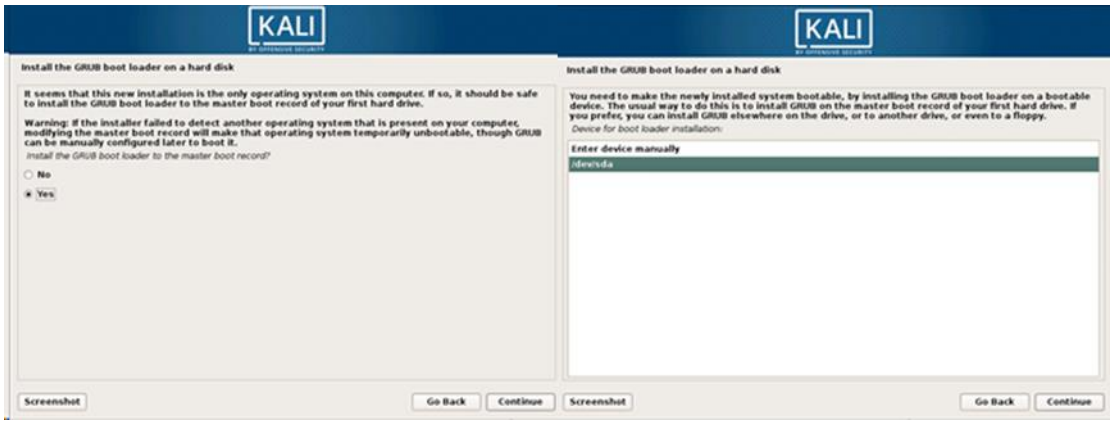


Imagen 25: Configuración e instalación del GRUB

Después de varias cargas e instalaciones, saltará una ventana avisando el término de la instalación, proceda a finalizar presionando continuar, el sistema se reiniciará y mostrará la pantalla de inicio de Kali Linux, imagen 26.

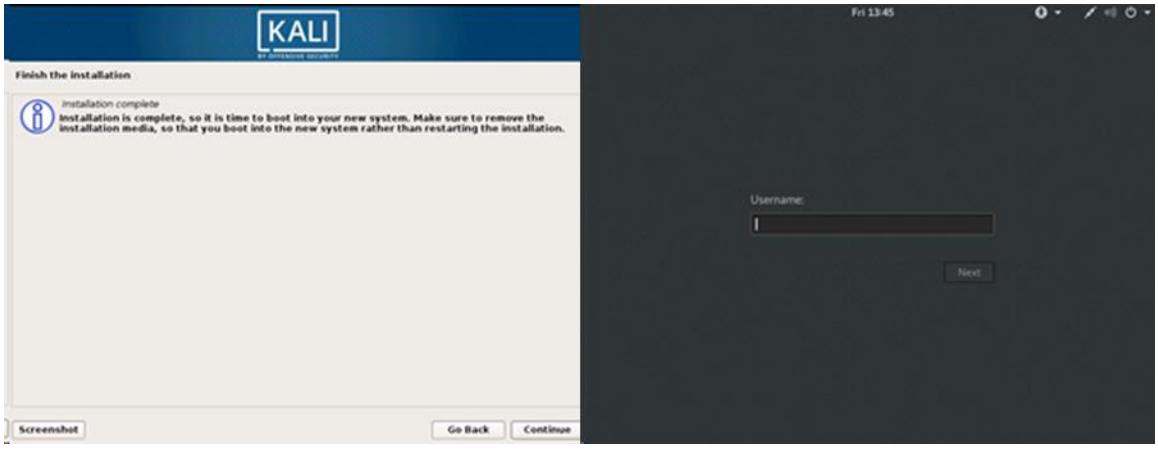


Imagen 26: Finalización de la instalación y pantalla de inicio de Kali Linux

Listado de Ilustraciones.

Ilustración 1: Ventana de Vulnerabilidad ____ 10
Ilustración 2: Evaluación de una amenaza; OWASP Top 10-2017[3] ____ 12
Ilustración 3: Interacción de un Framework Back-End y Front-End ____ 22
Ilustración 4: Modelo, Vista, Controlador. MVC ____ 23
Ilustración 5: Diagrama de un ataque cibernético ____ 23
Ilustración 6: Zed Attack Proxy Project_consola de pruebas ____ 34
Ilustración 7: Stand-Alone [12] ____ 34
Ilustración 8: Intercepting Proxy [12] ____ 34
Ilustración 9: Consola del proxy vista, edición y envío de mensajes de Burp Suite ____ 36
Ilustración 10: Emparejamiento y reemplazo de reglas [13]. ____ 36
Ilustración 11: Mapa del sitio objetivo [13]. ____ 37
Ilustración 12: Envío de respuestas y peticiones ____ 38
Ilustración 13: BApp Store [13]. ____ 38
Ilustración 14: Estructura RMVC en Laravel ____ 40
Ilustración 15: Estructura interna de las carpetas de Laravel ____ 41
Ilustración 16: Ejemplo de comando artisan en consola. ____ 42
Ilustración 17: Campo de Token CSRF ____ 43
Ilustración 18: Middleware ____ 44
Ilustración 19: Ejemplo de cifrado Hash ____ 45
Ilustración 20: Cifrado Hash en Laravel ____ 45
Ilustración 21: Archivo de la función Bycript en Laravel ____ 46
Ilustración 22:Archivo de la función Eloquent ORM en Laravel ____ 47
Ilustración 23: Generación de la llave en Laravel ____ 48
Ilustración 24: Integración de la plataforma, componentes y software. ____ 49
Ilustración 25: RMVC de la aplicación web creada ____ 50
Ilustración 26: Rutas definidas de la aplicación web ____ 51
Ilustración 27: Carpeta de vistas de la aplicación web ____ 51
Ilustración 28: Envío de mensajes de error de la aplicación web ____ 52
Ilustración 29: Pantalla de la vista registro de la aplicación ____ 52
Ilustración 30: Envío de mensajes de error de la aplicación ____ 53
Ilustración 31: Pantalla de la vista de login de la aplicación ____ 53
Ilustración 32: Implementación del campo CSRF_FIELD ____ 54
Ilustración 33: Vista de validación de usuario ____ 54
Ilustración 34: Definición de la base de datos clientes en el archivo .ENV ____ 54
Ilustración 35: Migración para la creación de la tabla users ____ 55
Ilustración 36: Tabla users dentro de la base de datos ____ 55
Ilustración 37: Función del método Login en el controlador ____ 56
Ilustración 38: Función del método Store en el controlador ____ 57
Ilustración 39: Registros realizados y almacenados en la base de datos, contraseñas cifradas ____ 57
Ilustración 40: Log de accesos a la aplicación web ____ 58
Ilustración 41: Envío de error 404 en excepciones. ____ 58
Ilustración 42: Mapeo de la aplicación web con Burp Suite ____ 60

Ilustración 43: Mapeo de la aplicación web con ZAP _____ 60
Ilustración 44: Ejemplo de configuración de rutas vulnerable _____ 61
Ilustración 45: Configuración de rutas de la aplicación web _____ 62
Ilustración 46: Formulario generado por el Spider _____ 62
Ilustración 47: Ejemplo de un registro utilizando un bot _____ 62
Ilustración 48: Iteraciones realizadas por el Spider _____ 63
Ilustración 49: Ejemplo de una captura de información por Man in the middle _____ 63
Ilustración 50: Prueba de un generador de contraseñas por fuerza bruta _____ 64
Ilustración 51: Token de sesión de Laravel interceptado _____ 64
Ilustración 52: Ejemplo de inicio de sesión con Token creado usuario identificado _____ 65
Ilustración 53: Ejemplo de inicio de sesión sin Token _____ 65
Ilustración 54: Plugin CO2 para SQL Injection _____ 66
Ilustración 55: Ejemplo de prueba fallida SQL Map _____ 66
Ilustración 56: Herramienta Repeater de Burp Suite _____ 67
Ilustración 57: Ejemplo de desempaquetado de Token – Burp Suite Decoder _____ 67
Ilustración 58: Prueba de cadenas vacías y datos aleatorios _____ 68
Ilustración 59: Prueba de usuarios no válidos _____ 68
Ilustración 60: Pruebas de cadenas vacías y datos aleatorios _____ 69
Ilustración 61: Registro con correo repetido _____ 69
Ilustración 62: Prueba de cadenas vacías y datos aleatorios _____ 69
Ilustración 63: Explotación por Clickjacking usando Burp Clickbandit _____ 70
Ilustración 64: Añadiendo campos de clickjacking a la aplicación _____ 71
Ilustración 65: Página resultante después de la post explotación con Burp Clickbandit _____ 71
Ilustración 66: Mensaje de alertas enviadas por el escáner de ZAP _____ 72
Ilustración 67: Evaluación de riesgos [14]. _____ 73
Ilustración 68: Ejemplo de cabeceras de seguridad Laravel _____ 76
Ilustración 69: Plan de auditoría preventivo _____ 80
Ilustración 70: SDLC [14]. _____ 81

Bibliografía

Alonso Caballero (2018). Hacking con Kali Linux

Brian Hatch & James Lee (2003), Hackers en Linux. Secretos y Soluciones para la Seguridad del Linux 2 Edición. España. Mcgraw Hill

CISCO Network Academy (2019). Indroduction to Cybersecurity español 0119 [1]

Christopher Pecoraro (2015). Mastering Laravel, Packt Publishing

Christopher Negus (2015) Linux Bible Eighth Edition, John Wiley & Sons, Inc

CVE. (2018). Common Vulnerabilities and Exposures https://www.cvedetails.com [6]

El economista (2018). Aseguradora AXA sufre ciberataque 2018 https://www.eleconomista.com.mx/sectorfinanciero/Aseguradora-AXA-sufre-ciberataque-20181024-0022 [9]

Foundation O. (2019). OWASP ZAP Getting Started Guide-2.6 [12]

Foundation O. (2018). OWASP Testing Guide V4, pag.11 [2] [14]

Foundation O. (2017). OWASP Top 10 – 2017 The Ten Most Critical Web Application Security Risk, pag.5[3] [15]

Foundation O. (2018). OWASP Risk Rating Methodology [16]

Foundation O. (2018). OWASP https://www.owasp.org

Gene Spafford, Simson. Garfinkel. (2001). Web Security, Privacy & Commerce, 2nd Edition. O'Reilly Media, Inc. [10]

Jennifer Niederst (2012). Learning Web Design Fourth Edition, Canada, O´Reilly Media, Inc

Matt Stauffer (2016) Laravel: Up and Running. A Framework for Building Modern PHP Apps, O´Reilly Media

Marting Bean (2015) Laravel 5 Essentials, Packt Publishing

Offensive Security OSCP (2019). What is kali linux https://docs.kali.org/introduction/what-is-kali-linux

Peter Kim (2015). The Hacker Playbook 2: Practical Guide to Penetration Testing Secure Planet LLC

PortSwigger Web Security (2019) Burp Suite https://portswigger.net/burp [13]

Sucuri (2017). Sucuri Hacked Report 2017 [4] [5]

Verizon (2017) 2017 Data Breach Investigations Report [7]

Imperva (2017). The state of web application vulnerabiliteies in 2017 https://www.imperva.com/blog/the-state-of-web-application-vulnerabilities-in-2017 [8]