ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

PROPUESTA DE UN MÉTODO DE MEJORES PRÁCTICAS PARA OPTIMIZAR EL NIVEL DE SEGURIDAD EN EL DESARROLLO DE SERVICIOS WEB RESTFul

MILTON FABIAN VILLA ESCUDERO

Trabajo de Titulación modalidad: Proyectos de Investigación y Desarrollo, presentado ante el Instituto de Posgrado y Educación Continua de la ESPOCH, como requisito parcial para la obtención del grado de:

MAGISTER EN SEGURIDAD TELEMÁTICA

RIOBAMBA – ECUADOR

Octubre 2019

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

CERTIFICACIÓN:

EL TRIBUNAL DE TRABAJO DE TITULACIÓN CERTIFICA QUE:

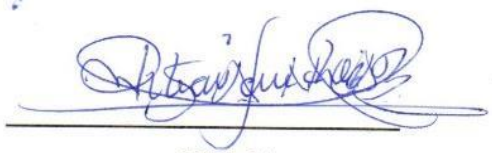
El Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, denominado: **"Propuesta de un método de mejores prácticas para optimizar el nivel de seguridad en el desarrollo de Servicios Web RESTFul"**, de responsabilidad del Sr. Milton Fabián Villa Escudero, ha sido minuciosamente revisado y se autoriza su presentación.

Tribunal:

Ing. Oswaldo Martínez, M.Sc.
PRESIDENTE

FIRMA

Ing. Wilian Xavier Sánchez Labré, M.Sc.
DIRECTOR DE TESIS



FIRMA

Ing. Iván Mesías Hidalgo Cajo, M.Sc.
MIEMBRO DEL TRIBUNAL

FIRMA

Ing. Saul Yasaca Pucuna, M.Sc.
MIEMBRO DEL TRIBUNAL

FIRMA

Riobamba, Octubre 2019

DERECHOS INTELECTUALES

Yo, Milton Fabián Villa Escudero, declaro que soy responsable de las ideas, doctrinas y resultados expuestos en el **Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo**, y el patrimonio intelectual del mismo pertenece a la Escuela Superior Politécnica de Chimborazo.

MILTON FABIAN VILLA ESCUDERO
No. Cédula 060371857-8

DECLARACIÓN DE AUTENTICIDAD

Yo, Milton Fabián Villa Escudero, declaro que el presente Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, es de mi autoría y que los resultados del mismo son auténticos y originales. Los textos constantes en el documento que provienen de otra fuente están debidamente citados y referenciados.

Como autor, asumo la responsabilidad legal y académica de los contenidos de este proyecto de investigación de maestría.

Riobamba, Octubre de 2019

MILTON FABIAN VILLA ESCUDERO
No. Cédula 060371857-8

DEDICATORIA

Este trabajo va dedicado a toda mi familia de manera especial a mi esposa Jeanneth y a mi hijo Alejandro *"My Champion",* que son pilares fundamentales en mi vida, mi gran apoyo e inspiración para superarme día a día, sobre todo por su comprensión y paciencia en los días que tomo en realizar la investigación, y también a mis padres que siempre creyeron en mí.

Fabián.

AGRADECIMIENTO

Quiero expresar mi agradecimiento primeramente a Dios por permitirme culminar con éxito los estudios de la maestría, a la Escuela Superior Politécnica de Chimborazo por haber hecho posible que el programa se desarrolle de la mejor manera, a los docentes que compartieron sus experiencias y enseñanzas.

Mi gratitud a los profesionales que fueron parte esencial para el desarrollo de esta investigación de manera especial al tutor y miembros de tribunal.

Fabián

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICOS

RESUMEN

El objetivo de la investigación fue proponer y aplicar en un escenario un método de mejores prácticas para el desarrollo de Servicios Web RESTFul, y fue desarrollado en base al análisis de métodos y técnicas existentes especificados en REST Security Cheat Sheet de OWASP. Se implementó dos escenarios con un aplicativo RESTFul cada uno implementado con Expressjs y Mongodb como gestor de datos, en donde se expuso un servicio implementado con los métodos existentes, *Escenario 1*, publicado en un servicio en la nube llamado Heroku con los valores de configuración que el servicio presta, y el *Escenario 2,* se implementó con el método propuesto que contiene las mejores prácticas de desarrollo de estos servicios y se publicó en un droplet (Servidor Virtual en la Nube) de Digital Ocean, donde se configuró el sistema operativo y las herramientas necesarias para el despliegue del aplicativo, además la configuración de un Certificado Digital junto con su propio subdominio. Los dos aplicativos están publicados y configurados en la web para poder acceder a ellos por medio de un cliente Postman y la herramienta pentesting Vooki.

Dado la hipótesis planteada la aplicación de una propuesta de un método de mejores prácticas optimizará el nivel de seguridad en el desarrollo de Servicios Web RESTFul, aplicando la observación en base a los parámetros a evaluar se obtuvo un 88.89% de optimización en el nivel de satisfacción y aplicando la herramienta pentesting Vooki se obtuvo un 90% de optimización en número de vulnerabilidades detectadas en autenticación y sesión iniciada se concluye que el método propuesto optimiza el nivel de seguridad en el desarrollo de estos servicios y se recomienda la adecuada configuración de la infraestructura donde va a trabajar el servicio.

Palabras clave: <TECNOLOGÍA Y CIENCIAS DE LA INGENIERÍA>, <INGENIERÍA DE SOFTWARE>, < SERVICIOS RESTful >, <SERVICIOS WEB>, <VOOKI (HERRAMIENTA)>, <SEGURIDAD TELEMATICA>, <PROTOCOLO HTTP>, <JSON (ESTRUCTURA)>, < SEGURIDAD JWT >, <MEJORES PRACTICAS DE DESARROLLO>

ABSTRACT

The main aim of this research was to propose and apply in a given scenario a better practice method for developing WEB RESTFul Services, and it was carried out on the basis of analysis methods and specified existing techniques in REST Security Cheat Sheet from OWASP. Two scenarios were implemented with a RESTFul application on each one implemented with Expressjs and Mongodb as data manager, where it was demonstrated an applied service with the existing methods, Scenario 1, posted on a cloud service called Heroku with the configuration values rendered by the service, Scenario 2, it was applied with the proposed method which has the better practices in the development stage of these services and it was published in a droplet (Virtual Cloud server) digital ocean, where the operating system was configured and the elementary tools in order to release the application, Furthermore a digital certificate configuration along with its own sub dominion. Both applications are published and configured on the web in order to be able to access to these applications by means of a Postman client and the pentesting Vooki tool.

Given the stated hypothesis a proposal application for a better practice method will improve the security level in the development of Web RESTFul services, by applying the analysis based on the parameters to be assessed,a 88.89% was obtained according to the optimization in the satisfaction degree and implementing pentesting Vooki tool a 90% optimization was gathered in a number of vulnerabilities detected on authenticity standards and started session. It is concluded that the proposed method to optimize the security level in these services development and it is suggested the appropriate configuration in thee framework where this service is going to be used.

KEYWORDS: < TECHNOLGY AND INGENEERING SCIENCES>, <SOFTWARE ENGINEERING>, <RESTful SERVICES>, <WEB SERVICES>, <VOOKI (TOOL)>, <TELEMATIC SECURITY>, <HTTP PROTOCOLE>, <JSON (STRUCTURE)>, <JWT SECURITY>, <BETTER DEVELOPMENT PRACTICES>.

CAPÍTULO I

1. INTRODUCCIÓN

1.1. Antecedentes

En la actualidad, la tendencia de las empresas en tener sus aplicativos integrados y orientadas a la globalización, han surgido nuevas arquitecturas y metodologías que pretenden brindar respuestas rápidas y optimas apoyando el concepto de aplicaciones distribuidas.

Los servicios, que son la parte base de dichas aplicaciones, son sistemas que proveen de una funcionalidad particular y están por lo general disponibles para permitir la integración con otros sistemas permitiendo la interoperabilidad y la globalización de la información.

Los Servicios Web RESTFul permiten una interconectividad con escaso consumo de servicios y poseen una arquitectura de escalabilidad, flexibilidad y portabilidad ya que puede funcionar independientemente en un servidor distinto a su Frontend.

El incremento en la implementación de estos servicios ha dado paso a un aumento en potenciales agujeros de seguridad lo que significa que los desarrolladores no solo deben concentrarse en la funcionalidad y agilidad de los servicios, sino que también deben entender el riesgo de mantener segura la información que procesan e intercambian.

La presente investigación pretende analizar los métodos de desarrollo habituales para la implementación de estos servicios, observar vulnerabilidades en estos servicios con toda esta información proponer un método de mejores prácticas aplicada al óptimo desarrollo de estos Servicios Web RESTFul tomando en cuenta los lineamientos de seguridad en su implementación.

1.2. Problema de Investigación

1.2.1. Planteamiento del Problema

Uno de los mecanismos actualmente predominante para el intercambio de información en el internet son los servicios web. Por tal razón surgen los servicios web permitiendo dicha interoperabilidad, independientemente de la plataforma en que sus aplicaciones estén implementadas. De entre los cuales muy comúnmente se utilizan los basados en API REST, compañías denominadas grandes como Google y Amazon ya han adoptado esta tecnología para implementarla en sus aplicativos.

Con los años, se desarrollaron más y más servicios web basados en el estilo arquitectónico REST, que utiliza la funcionalidad existente del protocolo de capa de aplicación Hypertext Transfer Protocol (HTTP). Esto resulta en un interés creciente en comparación con los servicios web tradicionales con el Protocolo simple de acceso a objetos (SOAP).

También las grandes empresas, como Twitter o Amazon, están utilizando interfaces similares a REST para sus servicios, que se muestran en sus documentos de la Interfaz de programación de aplicaciones (Giessler, Gebhart, Sarancin, Steinegger y Abeck, 2015).

Entre las características de RESTFul que se puede mencionar es el formato utilizado para el intercambio de la información dejando de lado el XML, y reemplazándolo por JSON (Javascript Object Notation), que es un formato estándar abierto que utiliza texto claro para facilitar el transporte, procesamiento e interoperabilidad durante la serialización y deserialización de la información, a través de servicios heterogéneos y aplicaciones y escrito en una multiplicidad de lenguajes de programación (Santos y Serrao, 2016).

Los servicios web RESTFul son sencillos, escalables y de fácil uso, pudiendo ser consumidos por una gran variedad de clientes, entre los que se pueden citar aplicativos webs, aplicativos de escritorio y aplicativos móviles. Las aplicaciones implementadas para dispositivos móviles particularmente necesitan comunicarse comúnmente con otros componentes del sistema por medio de servicios útiles y livianos, para que el procesamiento de la información sea óptimo con sus propios recursos de esta manera a menudo se utilizan los servicios Web RESTFul.

RESTFul posee una interfaz homogénea. Los recursos son utilizados por los cuatro métodos básicos de HTTP que son: GET, PUT, POST y DELETE. Además, hay dos métodos más adicionales, como HEAD y OPTION. Los metadatos utilizados para el consumo de los servicios se pueden mostrar por medio del método HEAD. El método de OPTION es para comprobar si los métodos están disponibles. Varios objetos utilizan la misma interfaz de control por la función de la interfaz homogénea (Arcuri, 2019).

El uso público de las APIs críticas como RESTFul abre la posibilidad de que bugs de software sean explotados de forma malintencionada. Por lo tanto, la identificación de las vulnerabilidades en los servicios web es un punto importante a ser tomado en cuenta en el desarrollo de nuevos aplicativos a implementar de manera segura (Segura, Parejo, Troya y Ruiz-Cortés, 2018).

Durante mucho tiempo se creyó que los servicios RESTFul deberían usarse para la integración adhoc en la Web, mientras que los servicios Big Web eran preferibles en escenarios de integración de aplicaciones empresariales con mayor esperanza de vida y requisitos de seguridad avanzados.

Sin embargo, hoy encontramos que cada vez más soluciones corporativas, incluso las más exigentes en seguridad, como los sistemas financieros y las operaciones de datos confidenciales, se basan en Servicios RESTFul o REST. A diferencia de los servicios de Big Web, no existe un marco de seguridad formal para los Servicios RESTFul, si bien Big Web admite SSL (al igual que REST), también es compatible con WS-Security, que agrega algunas características de seguridad empresarial (Lee, Jo, y Kim, 2015).

1.3. Formulación del Problema

¿La creación de una propuesta de un método de mejores prácticas permitirá optimizar el nivel de seguridad en el desarrollo de los Servicios Web RESTFul?

1.4. Sistematización del Problema

- ¿Cómo se implementan actualmente los Servicios Web RESTFul?
- ¿Cuáles son las vulnerabilidades que actualmente presentan los Servicios Web RESTFul?
- ¿Qué herramientas permiten medir el nivel de seguridad en Servicios Web RESTFul implementados?
- ¿Cómo ayudaría la creación de una guía con mejores prácticas en el desarrollo de Servicios Web RESTFul en cuanto a la seguridad en el intercambio de información?

1.5. Justificación de la Investigación

1.5.1. Justificación Teórica

Dado al creciente uso de las tecnologías para el intercambio de información, el servicio web es una aplicación de Internet que permite las interacciones entre los equipos encargados de entregar y recibir dicha información. Los servicios web actuales permiten a un proveedor de servicios publicar sus servicios disponibles en Internet, mientras que los clientes pueden buscar e invocar libremente estos servicios. Cuando se implementa un aplicativo web que gestione datos, este se

lo debería realizar siguiendo un conjunto de normas que garantice que la información que se procesa va a ser íntegra, confidencial y debe estar siempre disponible (Masood y Java, 2015).

El intercambio de información entre sistemas Web se lo realiza a través de servicios web. En RESTFul las conexiones realizadas entre servidores y clientes no crean sesiones por lo que los servidores no almacenan el estado de un cliente. Por lo tanto, los clientes tienen que expresar su estado actual a través de cabecera HTTP y el cuerpo del mensaje. Consecuentemente el proceso de autenticación del cliente, es necesario en REST cada vez que los servidores reciben solicitudes de los servicios (Yarygina, 2017).

Debido a esta particularidad, los Servicios REST están expuestos a tener sobrecargas de CPU y desgaste de recursos de red y sobre todo ataques a la integridad de la información trasmitida si no poseen una correcta definición de su arquitectura en el momento de la implementación (Prasher, 2018).

Los beneficios que se va a obtener tanto para investigadores expertos como novatos al seguir esta guía de buenas prácticas, es que se pretenderá sugerir la implementación de un Servicio Web RESTFul, el mismo que permitirá que la gestión de los datos que procesa sea más óptima y segura.

1.5.2. Justificación Metodológica

Para el reciente estudio se implementó el método científico que tiene su base y postura sobre la teoría mecanicista (todo es considerado como una máquina, que se dividó en partes pequeñas que permitieron estudiar, analizar y comprender sus nexos, interdependencia y conexiones entre el todo y sus partes) el cual se implementó en la parte investigativa.

La parte investigativa partió de las metodologías existentes para la creación de servicios RESTFul seleccionando las mejores prácticas de todas para formar la guía a proponer.

1.5.3. Justificación Práctica

Para la parte práctica demostrativa se desarrolló dos prototipos, el primero enfocado a implementar un servicio RESTFul considerando las principales características que debe tener y el segundo se realizó aplicando la guía propuesta de los Servicios Web RESTFul, cada uno implementado en una aplicación Web, en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible de un proyecto.

Para la medición del nivel de confianza se utilizan herramientas de monitoreo de tráfico en la red, herramientas de control de desarrollo seguro. El análisis de herramientas y los casos de estudio se realizan en un ambiente de pruebas implementado en un servidor de que posee dominio propio y certificado digital.

El ambiente de pruebas consta de una laptop que contendrá todas las herramientas de monitoreo instaladas y donde se realizará las peticiones será a un servidor real publicado en la web.

1.6. Objetivos

1.6.1. Objetivo General

Desarrollar una propuesta de un método de mejores prácticas para optimizar el nivel de seguridad en el desarrollo de Servicios Web RESTFul.

1.6.2. Objetivos Específicos

- Estudiar los métodos existentes de implementación para los Servicios Web RESTFul.
- Analizar las vulnerabilidades existentes en los Servicios Web RESTFul implementados.
- Diseñar una propuesta de un método de mejores prácticas para la creación de los Servicios Web RESTFul
- Medir el nivel de seguridad con herramientas específicas de los Servicios Web RESTFul implementados.

1.7. Hipótesis

La aplicación de la propuesta de un método de mejores prácticas permitirá optimizar el nivel de seguridad en el desarrollo de Servicios Web RESTFul

CAPÍTULO II

2. MARCO TEÓRICO

2.1. Antecedentes del Problema

Con la llegada de la era computacional en la nube, los servicios que se brindan en la web ha estallado en su capacidad de acceso a sus recursos en línea (Atlidakis, Godefroid y Polishchuk, 2018), facilitando el intercambio dinámico de información proporcionando la interoperabilidad entre un cliente y un servidor (Tarkowska et al., 2018).

Los Servicios Web RESTFul, en los últimos años se han presentado como una solución liviana para interconectar sistemas remotos en arquitecturas basadas en la nube convirtiéndose cada vez más populares. Sin embargo, los desarrolladores tienen que lidiar con una gran cantidad de recomendaciones y mejores prácticas, al ser éste un estilo arquitectónico en lugar de una especificación o un estándar, por lo tanto, el diseño adecuado de éstos servicios no es un proceso trivial (Ed-douibi, Cánovas, Gómez, Tisi y Cabot, 2016).

Esto implica que los desarrolladores deben tomar varias decisiones al momento de exponer sus servicios APIs, como que esquema, características, documentación o formatos deben tener éstos servicios, lo que puede dar como resultado diversos tipos de APIs y, en algunos casos, decisiones de diseño deficientes, por ejemplo, usar un solo verbo HTTP para recuperar y borrar un recurso (Neumann, Laranjeiro y Bernardino, 2018).

Estas decisiones también afectarían al desarrollador del lado del cliente, que debe adaptarse al estilo específico que se está utilizando e incluso puede afectar al proveedor, cuando se implementa un servicio que no se puede mantener a nivel de infraestructura (Neumann et al., 2018).

Las implementaciones de API públicas con información crítica, abren la posibilidad de que los errores de software se exploten de manera malintencionada ya que al igual que cualquier aplicación web vulnerable, estas API potencialmente fallidas representan una amenaza para la infraestructura empresarial y están expuestas a otras amenazas relacionadas, que anteriormente se pensaban como ataques centrados en aplicaciones web interactivas (Masood y Java, 2015).

Por lo tanto, la identificación de las vulnerabilidades en éstos servicios web, a través del análisis, se convierte en un área de investigación próspera e interesante en el mundo académico, la seguridad nacional y la industria (Masood y Java, 2015).

La seguridad no se tiene en cuenta de forma predeterminada en la arquitectura de Transferencia de Estado Representacional (REST), pero su arquitectura en capas brinda muchas oportunidades para implementarla (De Backere et al., 2014).

Actualmente, no ha habido una definición general sobre cómo el paradigma REST aborda la seguridad web, sin embargo, existen mecanismos de seguridad comunes, como la Seguridad de la capa de transporte (TLS), los objetos criptográficos en la Notación de objetos de JavaScript (JSON), la autenticación basada en token, la firma de solicitudes del lado del cliente, la autorización delegada y la autenticación compartida sin embargo es requerido un análisis avanzado para determinar cuál de éstos mecanismos se adhieren a las restricciones de estilo REST (Yarygina, 2017).

2.2. Bases Teóricas

2.2.1. Servicios API RESTFul

Las interfaces de programación de aplicaciones web (API) están proliferando rápidamente como un elemento clave para fomentar la reutilización, la integración y la innovación, lo que permite nuevos modelos de consumo de información, tales como aplicaciones móviles o de dispositivos inteligentes (televisión, relojes, etc) (Segura, Parejo, Troya y Ruiz-Cortés, 2018).

Una API es una forma inteligente de comunicación que proporciona una capa de seguridad, ya que los datos del cliente nunca están completamente expuestos al servidor y el servidor nunca está completamente expuesto al cliente, por lo tanto, un API es solo una interfaz para los servicios de back-end (Prasher, 2018).

Las API web suelen ser compatibles con el estilo arquitectónico de transferencia de estado de representación (REST), que se conoce como API Web RESTFul. Las API web RESTFul forman parte de un conjunto de los llamados Servicios Web RESTFul, donde cada servicio implementa uno o más operaciones CRUD (Crear, Recuperar, Actualizar, Eliminar) sobre un recurso (Segura et al., 2018).

REST es un patrón arquitectónico, específicamente diseñado para crear aplicaciones y servicios web que funcionan a través de la Internet pública (De Backere, et al, 2014). En la arquitectura REST, los datos y la funcionalidad se consideran recursos (Hamad, Saad, y Abed, 2010).

Los recursos dentro de la arquitectura REST se pueden manipular a través de un conjunto de Identificadores de Recursos Uniformes (URI) únicos lo que los hace direccionables y manipulables utilizando un protocolo de aplicación, generalmente HTTP (De Backere, et al., 2014).

La arquitectura REST es fundamentalmente una arquitectura cliente-servidor, y está diseñada para usar un protocolo de comunicación sin estado, habitualmente HTTP. En la arquitectura REST, los clientes y los servidores intercambian representaciones de recursos utilizando una interfaz y un protocolo estandarizados. Estos principios fomentan que las aplicaciones REST sean simples, ligeras y tengan un alto rendimiento (Hamad, Saad, & Abed, 2010).

El Protocolo de transferencia de hipertexto (HTTP) es un protocolo de aplicación para comunicaciones a través de una red, y es el principal protocolo de comunicación en la World Wide Web. Éste protocolo se define en una serie de documentos de tipo Requests for Comments (RFC) mantenidos por Internet Engineering Task Force (IETF) y el World Wide Web Consortium (W3C), como RFC 723010 y RFC 723111.

Un mensaje HTTP generalmente es enviado sobre TCP, y se compone de cuatro componentes principales:

Verbo / Método: es el tipo de operación a realizar, como obtener una página web específica.
Ruta de recursos (path): es un identificador para especificar en qué recurso se debe aplicar la operación HTTP, como la ruta de un documento HTML solicitado.
Encabezados (headers): son metadatos adicionales, expresados como una lista de pares clave / valor. Un ejemplo de metadatos es el encabezado de aceptación, que se usa para especificar el formato en el que se debe devolver el recurso, un recurso podría estar disponible en diferentes formatos (HTML, XML o JSON).
Cuerpo (body): es la carga útil del mensaje, como el texto HTML de una página web que se devuelve como respuesta a una solicitud realizada (Arcuri, 2019).

Los verbos son los métodos o acciones que están disponibles para interactuar con los recursos en el servidor. El número limitado de verbos en los sistemas REST confunde y frustra a los desarrolladores novatos en el enfoque. Lo que parecen ser restricciones arbitrarias e innecesarias, de hecho, están destinadas a fomentar un comportamiento predecible en formas no específicas de la aplicación. Hay cuatro verbos HTTP principales (***Tabla 1-2***), que son utilizados por sistemas RESTFul bien diseñados (Doelling, 2018).

Tabla 1-2: Tabla de verbos HTTP

Verbo	**Acción**
GET	Acceso de los recursos en modo lectura.
POST	Envía datos de un nuevo recurso al servidor para ser creado, por ejemplo, los valores de texto en un formulario web.
PUT	Envía datos que actualiza la información de un recurso específico almacenado.
DELETE	Elimina un recurso específico.

Fuente: (Arcuri, 2019)
Realizado por: Villa Fabián, 2019

Hay otros tres verbos que no se usan tan ampliamente pero que proporcionan cierto valor en su invocación (***Tabla 2-2***).

Tabla 2-2: Tabla de verbos HTTP no ampliamente usados.

Verbo	**Acción**
HEAD	Se utiliza para invocar un recurso sin recuperar realmente el recurso. Es una forma par que el cliente verifique la existencia del recurso.
OPTIONS	Se usa para averiguar a un servidor sobre un recurso al preguntar qué otros verbos son aplicables al recurso. Esto permite a los desarrolladores listar todas las operaciones HTTP disponibles en el recurso dado.
PARCHE	Realiza una actualización parcial del recurso dado, proporcionando una forma estandarizada de expresar actualizaciones parciales. Esto contrasta con PUT, donde el recurso se reemplaza completamente por uno nuevo.
CONNECTED	Establece una conexión de túnel a través de un proxy HTTP, generalmente necesario para las comunicaciones cifradas.

Fuente: (Doelling, 2018)
Realizado por: Villa Fabián, 2019

Cuando se produce una petición HTTP, el servidor devuelve una respuesta HTTP con encabezados y posiblemente una carga de información en el cuerpo (body). Esta respuesta

contendrá un código de estado numérico, de tres dígitos. Hay cinco grupos / familias de códigos, especificados por el primer dígito (Arcuri, 2019):

- 1xx: Informativo y solo se define en HTTP 1.1.
- 2xx: La solicitud fue correcta, presentación de contenido.
- 3xx: El recurso se movió de alguna manera a algún lugar.
- 4xx: La fuente de la solicitud hizo algo mal. Error de Cliente
- 5xx: El servidor se bloqueó debido a algún error en su código. Error de Servidor

Teniendo esto en cuenta, la **Tabla *3-2*** enumera algunos códigos de estado clásicos que una API podría usar.

Tabla 3-2: Tabla de errores HTTP.

Código	**Significado**
200	OK. La solicitud se ejecutó satisfactoriamente y el contenido solicitado fue devuelto.
201	Created. El recurso fue creado y reconocido por el servidor. En las respuestas a las solicitudes POST o PUT podría ser de utilidad. Al mismo tiempo, el recurso creado podría devolverse como parte del body de la respuesta.
204	No Content. La acción fue exitosa pero no devuelve contenido. Útil para acciones que no requieren un cuerpo de respuesta, como una acción DELETE.
301	Moved Permanently. Este recurso devuelve la ubicación de un recurso que se movió a otra ubicación. Esto es especialmente útil cuando las URL cambian con el tiempo tal vez debido a una migración, un cambio en la versión o algún otro cambio amenazante, mantener las antiguas y devolver una redirección a la nueva ubicación del recurso permite a los clientes antiguos actualizar sus referencias.
400	Bad request. La solicitud emitida tiene problemas debido a que puede faltar algunos parámetros requeridos en la solicitud del recurso. Un buen complemento de una respuesta 400 podría ser un mensaje de error para que un cliente pueda corregir la solicitud.
401	Unauthorized. Cuando en la solicitud de un recurso el usuario no puede acceder al recurso solicitado, esto es especialmente útil para la autenticación del usuario en el sistema.
403	Forbidden. Cuando en la solicitud el recurso no es accesible, pero a diferencia de 401, cuando el usuario se autentique, esto no afectará la respuesta.

404	Not found. La URL proporcionada no identifica a ningún recurso. Un buen complemento a esta respuesta podría ser un conjunto de URL válidas que el cliente puede usar para volver a encarrilarse en sus solicitudes.
405	Method not allowed. El verbo HTTP utilizado en un recurso que no está permitido. Esto puede suceder cuando al hacer un PUT sobre un recurso, este es de solo lectura.
500	Internal server error, Un código de error genérico cuando se cumple una condición inesperada y el servidor se bloquea. Normalmente, esta respuesta va acompañada de un mensaje de error que explica qué salió mal.

Fuente: (Doglio, 2015)
Realizado por: Villa Fabián, 2019

De esta manera, si se aplica los principios REST correctamente, los verbos HTTP deben usarse como se muestra en a continuación (***Figura 1-2***):

HTTP verb	Action	Response status code
GET	Request an existing resource	"**200 OK**" if the resource exists, "**404 Not Found**" if it does not exist, and "**500 Internal Server Error**" for other errors
PUT	Create or update a resource	"**201 CREATED**" if a new resource is created, "**200 OK**" if updated, and "**500 Internal Server Error**" for other errors
POST	Update an existing resource	"**200 OK**" if the resource has been updated successfully, "**404 Not Found**" if the resource to be updated does not exist, and "**500 Internal Server Error**" for other errors
DELETE	Delete a resource	"**200 OK**" if the resource has been deleted successfully, "**404 Not Found**" if the resource to be deleted does not exist, and "**500 Internal Server Error**" for other errors

Figura 1-2: Verbos Acciones y códigos de respuestas
Fuente: (Bojinov, 2015)

En la actualidad, uno de los principales mecanismos que se utiliza para el intercambio de información entre diferentes Servicios Web utiliza la notación de objetos de Javascript (JSON) sin embargo los recursos también se pueden representar utilizando diferentes formatos, como XML o XHTML (Santos y Serrao, 2016).

JSON es un formato estándar abierto de intercambio de datos liviano y legible por humanos, compuesto por pares de propiedad-valor (Segura et al., 2018), utiliza texto plano para facilitar el transporte, procesamiento e interoperabilidad durante la serialización y deserialización de la información a través de múltiples servicios y aplicaciones heterogéneas (Santos y Serrao, 2016).

JSON es una forma natural de representar datos (***Figura 2-2***), permitiendo la interoperabilidad entre servicios que se ejecutan en diferentes plataformas y que pueden ser consumidos por diferentes lenguajes de programación y se usa ampliamente para admitir la comunicación entre múltiples API disponibles en los servicios WWW REST (Santos y Serrao, 2016).

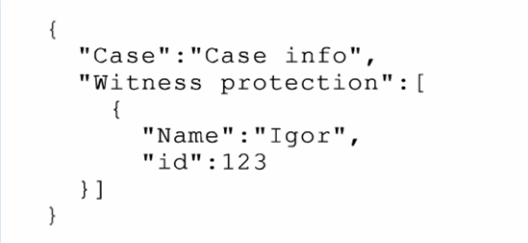


Figura 2-2: Modelo de estructura de JSON
Fuente: (Santos y Serrao, 2016)

Los valores de datos que JSON pueden incluir y pueden ser objetos (delimitados con corchetes), matrices (delimitados con corchetes) y referencias a otros URI, que permiten navegar de un recurso a otro (Segura et al., 2018) de esta manera, podría surgir un problema de seguridad en JSON que cobra impulso debido a las sensibles características de la información que se transporta y está encapsulada entre este ecosistema heterogéneo distribuido (Santos y Serrao, 2016).

2.3. Mecanismos de Seguridad en Servicios Web RESTFul

Las API RESTFul son igualmente vulnerables a los ataques comunes dirigidos a aplicaciones web. Sobre la base de similitudes, se puede inferir fácilmente, que se espera que los Servicios REST sufran los mismos problemas de seguridad similares a los de sus homólogos web, sin embargo, no obtienen el mismo nivel de reconocimiento debido a la limitada exposición pública ya que se pueden invocar directamente desde el navegador debido a la relajación de los controles de nivel medio (Masood y Java, 2015).

El Servicio Web RESTFul, lucha constantemente con el proceso de autenticación de los usuarios. Sin embargo, hay problemas de autenticación debido a que REST no utiliza sesiones entre el servidor y el cliente. Por lo tanto, el proceso de autenticación del cliente es necesario para REST cuando los servidores reciben las solicitudes de los clientes.

Un proceso de autenticación de usuario frecuente puede provocar una sobrecarga de la CPU o desperdiciar una gran cantidad de recursos de red. Los servidores tienen que autenticar la información del cliente para cada solicitud. En lugar de enviar la identificación y la contraseña a

través del método de publicación, actualmente existe los siguientes tipos de autenticación implementadas para REST (Lee, Jo, y Kim, 2015).

Autenticación básica HTTP, utiliza el ID y la contraseña de un cliente para autenticar la solicitud del cliente en el encabezado HTTP. La ***Figura 3-2*** muestra las fases de autenticación HTTP. Cuando un servidor solicita una autenticación para un cliente, el servidor envía un mensaje "HTTP 401 No autorizado". El ID y la contraseña del cliente se codifican con Base64 y se almacenan en el encabezado de autenticación.

Como no están cifrados o con hash, generalmente se envían a través de HTTPS o SSL. Sin embargo, este método tiene un problema crítico que no admite una función de cierre de sesión. Debido a la necesidad de guardar las credenciales en el encabezado HTTP, la autenticación básica HTTP puede estar expuesta a ataques de repetición, ataque de inyección y secuestro de middleware.

Figura 3-2: Modelo de autenticación básica
Fuente: (Bojinov, 2015)

Autenticación HTTP Digest, es una versión avanzada de la autenticación básica HTTP. La autenticación HTTP Digest cifra el ID y la contraseña del cliente a través de un hash como MD5. Al crear un nonce (número que solo se puede usar una vez) en el lado del cliente, puede proteger el hash de un ataque de Rainbow Table. Además, la marca de tiempo creada en un servidor puede proteger el mensaje de un cliente de un ataque de reproducción.

Sin embargo, la autenticación HTTP Digest tiene varias vulnerabilidades con respecto a la seguridad. Dado que no proporciona un método de servidor de confirmación al cliente, la autenticación de compendio de HTTP puede ser atacada por un ataque Man-in-the-Mididdle que puede cambiar la autenticación a la autenticación básica. En otras palabras, el propósito mismo de la autenticación HTTP Digest puede ser anulado.

Autenticación basada en token (OAuth), utiliza un token en lugar del ID y la contraseña del usuario. La ***Figura 4-2*** muestra la forma en que funciona la autenticación basada en token cuando se autentica la información de un usuario. Resource Server (RS) redirige el navegador del usuario a PKG cuando un usuario solicita un servicio RS. En primer lugar, los usuarios inician sesión en el Servidor de autorización (AS) para obtener un token.

En segundo lugar, RS obtiene el token del AS. Por último, RS verifica los mensajes del usuario que utiliza el token. En consecuencia, debido al uso de un token en la comunicación entre un usuario y RS, la identificación y las contraseñas de los usuarios están protegidas de un tercero. Debido a esta ventaja, el método OAuth se implementa con frecuencia por varias compañías de servicios web.

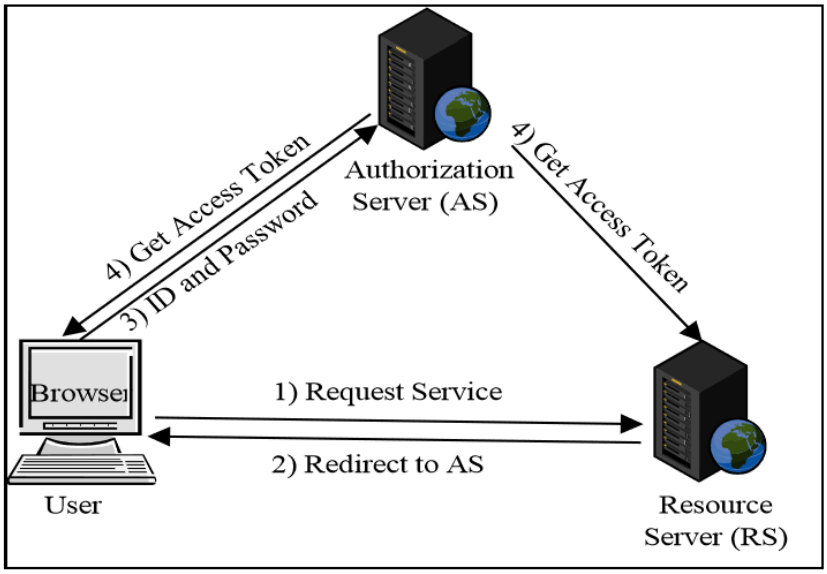


Figura 4-2: Autenticación basada en token
Fuente: (Lee et al., 2015)

Debido a los problemas de autenticación de estos dos métodos, la mayoría de las empresas utilizan OAuth en lugar de la autenticación básica HTTP y la autenticación Digest HTTP. Sin embargo, OAuth también es vulnerable a violaciones de seguridad, es decir, vulnerabilidad de redirección encubierta (CRV).

Autenticación OAuth2.0, es el protocolo estándar de la industria para la autorización. El marco de autorización OAuth 2.0 permite que una aplicación de terceros obtenga acceso limitado a un servicio HTTP. En esta autenticación, el usuario todavía tendrá que enviar el nombre de usuario y la contraseña en el cuerpo de la solicitud de manera similar a la autenticación básica, pero también introducir tokens.

Los tokens se almacenarán y se mantendrán en el lado del servidor. Se puede utilizar el mismo token para llamar al servicio cualquier cantidad de veces hasta que caduque. Cuando el token de acceso ha caducado, el usuario puede utilizar refreshToken para obtener el nuevo acceso (Salibindla, 2018).

Autenticación de token web JSON (JWT), un token JWT es en realidad un objeto JSON completo que ha sido codificado en base64 y luego firmado con una clave compartida simétrica o usando un par de claves pública/privada.

El JWT puede contener dicha información, incluido el sujeto o el ID de usuario, cuándo se emitió el token y cuándo caduca. Al firmar con un secret (clave secreta), el JWT garantiza que solo el usuario definido puede generar un token único, que no puede abrirse para la manipulación (como la modificación de la ID de usuario o cuando caduca).

Sin embargo, aunque el JWT está firmado, los JWT generalmente no se cifran automáticamente (el cifrado JWT es una característica opcional). Esto significa que cualquier persona que tenga acceso al token puede leer cualquier información que esté en el token. Uno de los beneficios de los JWT es que se pueden utilizar sin una tienda auxiliar. Toda la información requerida para autenticar al usuario está contenida dentro del token (Salibindla, 2018).

2.3.1. Consideraciones de Seguridad utilizados en Desarrollo de Servicios Web RESTFul

Se necesitan mecanismos de seguridad adecuados para construir Servicios Web RESTFul seguros, como la Seguridad de la capa de transporte (TLS), los objetos criptográficos en la Notación de objetos de JavaScript (JSON), la autenticación basada en token, la firma de solicitudes del lado del cliente y la autorización delegada y la autenticación compartida.

TLS, fue diseñado originalmente para ser independiente de cualquier protocolo de aplicación y se ha convertido en un protocolo de seguridad de facto en la Web. Aunque el diseño de TLS admite la autenticación mutua, HTTPS en su forma actual se usa en gran medida para autenticar la puerta de enlace, pero no para el cliente.

Por lo tanto, la autenticación del cliente se debe proporcionar en el nivel de la aplicación (mensaje). Para proporcionar una mayor seguridad, así como la autenticación del cliente, TLS puede combinarse, a menudo, con cifrado y firma en el nivel de mensaje.

Las normas para objetos criptográficos en JSON y XML, que se pueden ver como contenedores que incorporan datos protegidos y la información necesaria para su procesamiento, que se crearon para abordar las necesidades de seguridad en el nivel del mensaje y para facilitar la interoperabilidad.

Autenticación basada en token, a través de cookies HTTP es el mecanismo de autenticación más ampliamente adoptado en aplicaciones web. Se basa en una noción de tokens de seguridad: objetos criptográficos que contienen información relevante para la autenticación o autorización.

Un token de autenticación es generado por un servicio web y enviado a un cliente para uso futuro, luego de la validación exitosa de las credenciales del cliente durante el inicio de sesión del usuario o una nueva autenticación.

Un token se puede ver como un reemplazo temporal de las credenciales del cliente: cada solicitud de un cliente debe incluir un token válido para que se ejecute.

Los tokens de seguridad creados por el servidor garantizan la escalabilidad de la solución y la falta de estado del servidor al transferir la responsabilidad de mantenimiento de los tokens a los clientes. Además, una vida útil limitada de tokens de seguridad los hace superiores al uso directo de contraseñas, como en HTTP Basic / Digest Authentication.

Firma de la solicitud del lado del cliente, y la protección contra manipulación indebida en tránsito al requerir que un cliente firme cada solicitud, se han implementado en muchos servicios REST. A través del uso de claves criptográficas que se establecen entre las partes durante o después del paso de autenticación inicial.

Varios servicios web importantes como Amazon Web Services (AWS) y Microsoft Azure han implementado con éxito la firma de solicitudes con encabezados HTTP. Ambos son servicios en la nube destinados solo para uso programático a través de REST APIs.

Las solicitudes firmadas por el cliente proporcionan una autenticación más fuerte que los simples esquemas basados en token. La firma de cada solicitud de cliente, mitiga de manera efectiva los ataques de secuestro de sesión al limitar el daño a una sola solicitud.
Una firma nunca deja a un cliente, lo que hace que robar la clave sea mucho más difícil que robar un token ya que no solo se almacena en el cliente, sino que también se envía repetidamente por el canal. Como suele suceder, una mayor seguridad tiene un precio de menor escalabilidad y mayor complejidad, ya que un servidor necesita mantener una clave separada para cada usuario.

Autorización Delegada y Autenticación Compartida, se han convertido en una parte integral de la seguridad web moderna. Los protocolos de seguridad populares que subyacen a estos métodos en su mayoría ya que ejemplifican la autenticación basada en token mencionada anteriormente, por lo tanto, comparten las ventajas y desventajas de la autenticación basada en token.

El deseo de separar el proceso de inicio de sesión en el servidor del proceso de otorgar permisos a una aplicación cliente en nombre del usuario ha estimulado la aparición de OAuth que es un protocolo de autorización delegada que proporciona a aplicaciones de terceros (clientes) con acceso delegado a recursos protegidos en nombre de un usuario (propietario del recurso).

OAuth 1.0 habilita la autenticación del cliente y la integración de mensajes, mientras que OAuth 2.0 se utiliza como una capa subyacente para los protocolos de autenticación compartidos y los sistemas de inicio de sesión único (SSO). Algunos ejemplos destacados son OpenID Connect, Facebook Login y Sign In en Twitter.

En tales esquemas, el usuario se autentica en un servicio de terceros (una Parte que confía o RP) utilizando una identidad digital en un Proveedor de Identidad (IdP) a elección del usuario. Los análisis de seguridad de las soluciones de SSO basadas en OAuth implementadas comercialmente (es decir, proveedores populares de inicio de sesión social) han revelado tener ciertos problemas de seguridad y privacidad (Yarygina, 2017).

2.3.2. Metodologías de Desarrollo RESTFul

De acuerdo con mejores prácticas y metodologías revisadas, en el Articulo *"Once consejos rápidos para construir una API REST utilizable para las ciencias de la vida"* (Tarkowska et al., 2018) se mencionan lineamientos a considerar para la implementación de estos servicios, tales lineamientos abarcan todos los puntos importantes expuestos en artículos relacionados por lo que se detallan a continuación:

Reduzca los costos de soporte al proporcionar una buena documentación. Incluso la API REST mejor diseñada y más intuitiva decaerá sin ser utilizada si no hay una documentación clara, completa y actualizada, preferiblemente con ejemplos del mundo real. Los lenguajes de descripción de API REST (DL) documentan las API en formatos humanos y legibles por máquina. Una DL líder es la especificación OpenAPI (OEA; originalmente conocida como Especificación

Swagger, https://swagger.io/), una especificación portátil y abierta que proporciona metadatos para API basadas en la arquitectura

Diseñe una API con URL estables, coherentes y claras. Un primer paso para crear una API REST es determinar las funcionalidades que su recurso web pretende proporcionar. REST impone que cada recurso debe ser direccionable de forma única y se puede acceder a él, mediante localizadores de recursos únicos (URL), denominados endpoints.

Los esquemas de URL que utilizan sustantivos como etiquetas que funcionan con los verbos HTTP proporcionan un control transparente de las acciones. Las direcciones URL legibles para las personas que siguen este estilo de esquema permiten recuperar fácilmente una colección de recursos o detalles de una sola entidad. Desarrollar un esquema de URL sensible asegurará que su API REST sea fácil de entender y usar.

Use encabezados HTTP estándar para influir en cómo los clientes manejarán su contenido. Una respuesta HTTP del servidor al cliente consta de dos partes, la información sobre cómo se procesó la solicitud, incluidos los encabezados HTTP con un código de estado, y Cuerpo del mensaje que contiene los datos del recurso. Los encabezados HTTP representan los metadatos de una respuesta.

Utilice los formatos de datos estandarizados apropiados. Las API REST pueden devolver datos en varios formatos, denominados tipos de medios, a través de un proceso llamado negociación de contenido. Los tipos y formatos de medios más utilizados son:

- application/json (JSON)
- application/xml
- texto/csv
- application/octet-stream

Utilice las respuestas HTTP estándar para influir en cómo los clientes manejarán su contenido. HTTP define un conjunto de verbos que se pueden aplicar a un endpoint para cambiar la acción realizada. Los más comúnmente utilizados son "GET" para transferir una representación actual del recurso desde un punto final. 'POST', 'PUT', 'PATCH' y 'DELETE' realizan una operación de procesamiento, que podría ser destructiva para los datos.

Permita que su API se use en otros sitios web al permitir el uso compartido de recursos de origen cruzado. Todos los navegadores web implementan la política del mismo origen, una medida de seguridad que permite que el código JavaScript realice solicitudes solo entre un

servidor y clientes del mismo origen, que evita que los códigos maliciosos secuestren datos privados, como las cookies.

Se dice que dos sitios web son del mismo origen si tienen un esquema idéntico (por ejemplo, https), host (por ejemplo, www.ebi.ac.uk) y puerto (por ejemplo, 443). El intercambio de recursos de origen cruzado (CORS), es utilizado automáticamente por un navegador cuando se realiza una solicitud de origen cruzado.

El navegador agregará un encabezado de Origen a una solicitud. Un servidor puede responder con un encabezado de Access-Control-Allow-Origin que indica el origen permitido o un "*', que indica que todos los orígenes están permitidos. Devolver el encabezado 'Access-Control-Allow-Origin' en cualquier solicitud 'GET', 'HEAD' o 'POST' en muchos casos es suficiente para habilitar CORS.

Ayude a los clientes a usar su API dándoles enlaces pre generados. Un concepto clave que sustenta la red mundial y HTML es la conciliación de la navegación de enlaces entre los clientes. Las API REST pueden devolver acciones con una respuesta e informar a los clientes de las acciones disponibles para ellos. Dado que estas palabras clave de acción siguen siendo coherentes con los cambios de API, el esquema de URL del servidor es libre de cambiar.

Autenticación mediante un método estándar. La autenticación es el proceso de identificación de un cliente cuando el servidor cuenta con un nombre de usuario y una contraseña (credenciales) que coinciden con la información de una persona autorizada dentro de un servicio de autenticación. Se recomienda a los desarrolladores de API a utilizar métodos de autenticación comunes como OAuth 2.0, JsonWebToken y autenticación básica a través de un protocolo cifrado como una conexión HTTPS (TLS 1.3) y a estar conscientes de los problemas de seguridad que pueden surgir.

Mantenga su API funcionando a toda costa. Una API útil es aquella que permanece disponible en todo momento, la clave es lograr un equilibrio entre brindar un servicio confiable y operar dentro de sus restricciones de servicio. Debido a que las API están diseñadas para uso programático, deben escalarse según la demanda y ser almacenadas en caché para evitar la reducción del tráfico de red y evitar los ataques no deseados de denegación de servicio.

Versión de su API y permita que los clientes migren en respuesta a sus cambios. El control de versiones es uno de los temas más debatidos entre los desarrolladores y usuarios de REST API,

y muchos eligen no versionar. Una API REST debe conservar el diseño de la URL y los formatos de datos para evitar que se "rompan" las implementaciones de los clientes.

Compruebe si un framework web puede ayudarlo. Muchos de los consejos mencionados anteriormente ya han sido implementados por varios frameworks compatibles con REST, en una variedad de lenguajes de programación, por ejemplo, Spring (Java); Django o Frasco (Python); Restify, hapiJS, Express o Loopback (Node.js); y Catalys o Mojolicious (Perl). Si su API REST se construye utilizando uno de estos frameworks, muchos de estos consejos ya estarán implementados o disponibles a través de un complemento para simplificar la implementación.

2.3.3. Consideraciones de OWASP para RESTFul seguros

De acuerdo a (REST Security Cheat Sheet de OWASP, 2019), a continuación, se expone los lineamientos recomendados para la implementación segura de estos servicios:

HTTPS

Los Servicios REST seguros solo deben proporcionar endpoints HTTPS. Esto protege las credenciales de autenticación en tránsito, por ejemplo, contraseñas, claves API o JSON Web Tokens. También permite a los clientes autenticar el servicio y garantiza la integridad de los datos transmitidos. Considere el uso de certificados del lado del cliente autenticados mutuamente para brindar protección adicional para servicios web altamente privilegiados.

Control de Acceso

Los Servicios REST no públicos deben realizar un control de acceso en cada enpoint de API. Los servicios web en aplicaciones monolíticas implementan esto mediante la autenticación de usuarios, la lógica de autorización y la administración de sesiones. La decisión de control de acceso debe ser tomada localmente por los endpoints REST y la autenticación del usuario debe estar centralizada en un proveedor de identidad (IdP), que emite tokens de acceso.

JWT

Al parecer que hay una convergencia hacia el uso de JSON Web Tokens (JWT) como el formato para los tokens de seguridad. Los JWT son estructuras de datos JSON que contienen un conjunto de notificaciones que se pueden usar para tomar decisiones de control de acceso. Se puede usar una firma criptográfica o un código de autenticación de mensaje (MAC) para proteger la integridad del JWT.

Si los MAC se utilizan para la protección de integridad, cada servicio que puede validar JWT también puede crear nuevos JWT con la misma clave. Esto significa que todos los servicios que utilizan la misma clave deben confiar mutuamente.

API Keys

Los Servicios de REST públicos sin control de acceso corren el riesgo de ser vulnerables. Las claves API se pueden utilizar para mitigar este riesgo. También son utilizados a menudo por la organización para monetizar las API; En lugar de bloquear llamadas de alta frecuencia, los clientes tienen acceso de acuerdo con un plan de acceso comprado. Las claves API pueden reducir el impacto de los ataques de denegación de servicio. Sin embargo, cuando se emiten a clientes externos, son relativamente fáciles de comprometer, se recomienda:

- Requerir claves de API para cada solicitud al punto extremo protegido.
- Devuelva el código de respuesta HTTP Demasiadas solicitudes 429 si las solicitudes llegan demasiado rápido.
- Revocar la clave API si el cliente viola el acuerdo de uso.
- No confíe exclusivamente en las claves de API para proteger recursos sensibles, críticos o de alto valor.

Restringir Métodos HTTP

Aplicar una lista blanca de métodos HTTP permitidos, p. Ej. GET, POST, PUT. Rechace todas las solicitudes que no coincidan con la lista blanca con el código de respuesta HTTP 405 Método no permitido. Asegúrese de que la persona que llama esté autorizada para usar el método HTTP entrante en la recopilación de recursos, la acción y el registro.

Validar Tipos de Contenido

Un cuerpo de solicitud o respuesta REST debe coincidir con el tipo de contenido deseado en el encabezado. De lo contrario, esto podría causar una mala interpretación en el lado del consumidor/productor y provocar la inyección/ejecución del código. Documente todos los tipos de contenido admitidos en su API.

Gestionar Endpoints

- Evite exponer la gestión de endpoints a través de Internet.
- Si gestión de endpoints deben ser accesibles a través de Internet, asegúrese de que los usuarios deben usar un mecanismo de autenticación fuerte, por ejemplo multifactor

- Exponga gestión de endpoints a través de diferentes puertos HTTP o hosts, preferiblemente en una NIC diferente y una subred restringida.
- Restrinja el acceso a estos endpoints mediante reglas de firewall o mediante el uso de listas de control de acceso.

Manejo de Errores

Responda con mensajes de error genéricos: evite revelar detalles de la falla innecesariamente. No pase detalles técnicos (por ejemplo, pilas de llamadas u otras sugerencias internas) al cliente.

Registros de Auditoría

- Escribir registros de auditoría antes y después de los eventos relacionados con la seguridad.
- Considere el registro de errores de validación de token para detectar ataques.
- Tenga cuidado de los ataques de inyección de registros al validar los datos de registro de antemano.

Seguridad de Encabezados

Para asegurarse de que el contenido de un recurso dado sea interpretado correctamente por el navegador, el servidor siempre debe enviar el encabezado Content-Type con el tipo de contenido correcto, y preferiblemente el encabezado Content-Type debe incluir un conjunto de caracteres. El servidor también debe enviar el encabezado de seguridad de Opciones-Tipo-X: nosniff para asegurarse de que el navegador no intente detectar un Tipo de Contenido diferente al que realmente se envía (puede llevar a XSS).

CORS

El intercambio de recursos de origen cruzado (CORS) es un estándar del W3C para especificar de manera flexible qué solicitudes de dominio cruzado están permitidas. Al entregar los encabezados CORS apropiados, las señales de la API REST en el navegador de los dominios, orígenes AKA, pueden realizar llamadas de JavaScript al servicio REST.

Deshabilite los encabezados CORS si las llamadas entre dominios no son compatibles/esperadas. Sea lo más específico posible y tan general como sea necesario al configurar los orígenes de las llamadas de varios dominios.

Información Sensible en Solicitudes HTTP.

Los Servicios Web RESTFul deben tener cuidado para evitar la pérdida de credenciales. Las contraseñas, los tokens de seguridad y las claves de API no deben aparecer en la URL, ya que esto puede capturarse en los registros del servidor web, lo que los hace intrínsecamente valiosos.

En las solicitudes POST/PUT, los datos confidenciales deben transferirse en el cuerpo de la solicitud o en los encabezados de las solicitudes. En las solicitudes GET, los datos confidenciales deben transferirse en un encabezado HTTP.

- Correcto:

https://example.com/resourceCollection/[ID]/action

https://twitter.com/vanderaj/lists

- Incorrecto:

https://example.com/controller/123/action?apiKey=a53f435643de32

porque la clave API está en la URL.

Código de retorno HTTP

HTTP define el código de estado. Al diseñar la API REST, no solo use 200 para el éxito o 404 para el error. Siempre use el código de estado semánticamente apropiado para la respuesta.

CAPÍTULO III

3. METODOLOGÍA DE INVESTIGACIÓN

3.1. Introducción

En esta sección se determina los procedimientos y/o técnicas utilizados, para definir el método de mejores prácticas para optimizar el nivel de seguridad en el desarrollo de los Servicios Web RESTFul, así como los escenarios de estudio, y las herramientas que se utilizaron.

3.2. Tipo y Diseño de la Investigación

3.2.1. Tipo de Investigación

La presente investigación se la considera de tipo descriptiva y aplicada, ya que se basa en conocimientos existentes derivados de la investigación, con la finalidad de aplicar técnicas conocidas de desarrollo de Servicios Web RESTFul y obtener un método de mejores prácticas para su desarrollo mitigando así los posibles riesgos de seguridad que comprometan la información gestionada en ellos.

3.2.2. Diseño de la Investigación

El diseño de la investigación es del tipo experimental en donde luego de analizar las 16 técnicas de desarrollo de Servicios Web RESTFul detalladas en OWASP, se definió un modelo de mejores prácticas para la implementación, y que fue evaluado con el uso de analizadores de vulnerabilidades sobre dos prototipos, el primero implementado con los lineamientos de desarrollo existentes y el otro prototipo con el método propuesto.

3.3. Métodos y Técnicas De Investigación

3.3.1. Métodos

Los métodos a que se utilizaron en esta investigación son:

Método Científico que propone un flujo de etapas a recorrer para obtener un conocimiento valido con el uso de instrumentos fiables, estas etapas son:

- Planteamiento del Problema
- Formulación de la Hipótesis
- Levantamiento de la Información
- Análisis e Interpretación de Resultados
- Comprobación de la Hipótesis
- Difusión de Resultados

Método Deductivo debido a al análisis realizado de las técnicas de desarrollo actuales de Servicios Web RESTFul se definió el método de mejores prácticas más adecuado para la implementación de estos servicios.

3.3.2. Técnicas

Las técnicas que se utilizaron en esta investigación son

- **Búsqueda de Información:** Permite obtener la información necesaria acerca del objeto de estudio de la investigación para su desarrollo, utilizando las fuentes secundarias disponibles.
- **Pruebas:** Realiza experimentos en escenarios de laboratorio.
- **Observación:** Permite determinar los resultados de las pruebas realizadas en los escenarios de laboratorio.
- **Análisis:** Determina los resultados de la investigación

3.4. Instrumentos

Para la investigación se ha optado por herramientas Open Source que permitieron la implementación de los escenarios y ejecutar las diferentes pruebas para la recolección de datos.

El uso de estas herramientas permitió levantar los ambientes de desarrollo adecuados para poder realizar la investigación sin inconvenientes.

3.4.1. Node JS

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos (***Figura 1-3***).

En la investigación será el lenguaje de programamos utilizado para la implementación de los aplicativos con el uso del framework express[1] versión 4 orientado para creación de API RESTFul.

Figura 1-3: Node JS
Fuente: (The Linux Foundation, s.f., 2019)

3.4.2. Mongo DB

MongoDB es un motor de base de datos NoSQL, orientada a documentos lo que quiere decir que almacena datos en documentos planos con una representación binaria de JSON denominada BSON, y no necesita de la definición de un esquema.

En la aplicación será el motor de base de datos (***Figura 2-3***) utilizado para gestionar la data que usaran los aplicativos.

Figura 2-3: Mongo DB
Fuente: (MongoDB, Inc., 2019)

3.4.3. Sublime Text 3

Es un editor de código multiplataforma ligero que permite el desarrollo de software de una manera ágil y sencilla con sus potentes helpers que hacen del desarrollo una experiencia agradable.

En la investigación sublime text será el IDE (***Figura 3-3***) de implementación de los aplicativos.

[1] Framework de nodejs. https://expressjs.com/

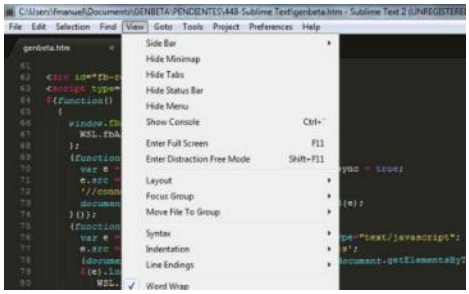


Figura 3-3*:* Sublime Text
Fuente: (Sublime HQ Pty Ltd, s.f., 2019)

3.4.4. Docker

Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software que permiten que varios de ellos se puedan ejecutar dentro de una misma instancia evitando la sobrecarga de mantener máquinas virtuales.

En la investigación docker (***Figura 4-3***) permitirá tener levantado el motor de base de datos MongoDB.

Figura 4-3*:* Doker
Fuente: (Docker Inc., s.f., 2019)

3.4.5. GitHub

GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código (***Figura 5-3***).

Figura 5-3*:* GitHub
Fuente: (GitHub, Inc., s.f., 2019)

3.4.6. Heroku

Es una plataforma de servicios en la nube que soporta múltiples lenguajes de programación y que permite a las empresas construir, entregar, supervisar aplicaciones y alojarlas en la nube además permite desplegar versiones, hacer rollback, gestionar dependencias.

Heroku (***Figura 6-3***) dispone los denominados add ons, gracias a los que se puede añadir funcionalidad extra a las aplicaciones de forma realmente sencilla, por ejemplo, memcached, redis, postgres, mongolab etc.

Figura 6-3*:* Heroku
Fuente: (Salesforce, s.f., 2019)

3.4.7. Digital Ocean

Es un servicio en la nube que está muy ligado al mundo de GNU/Linux y de la administración de sistemas. Además, es una de las compañías de «Cloud Hosting» que ha experimentado un gran crecimiento en los últimos años. DigitalOcean (***Figura 7-3***) maneja el concepto de Droplet (servidor virtual en la nube) para designar a cada uno de los servidores virtuales, exactamente servidores virtuales privados, los cuales ofrecen en alquiler.

Figura 7-3*:* Digital Ocean
Fuente: (DigitalOcean, LLC, s.f., 2019)

Para el análisis de vulnerabilidades de Servicios Web RESTFul se utilizaron herramientas que existen en el mercado y algunas son de libre uso Open Source y estas permitieron obtener un resultado más óptimo para el análisis posterior.

- **Postman:** es un cliente HTTP que permite gestionar las peticiones a nuestras API´s. Postman tiene muchas funcionalidades para gestionar todo el ciclo de vida de nuestra API. Postman es muy útil a la hora de programar y hacer pruebas, puesto que ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros desarrollos.
- **Vooki free web application vulnerability scanner:** Es un escáner de vulnerabilidades de aplicaciones web gratuito que brinda informes de escaneo perfectos sobre las redes y aplicaciones escaneadas. Es una herramienta fácil de usar que puede escanear fácilmente cualquier aplicación web y encontrar vulnerabilidades de seguridad. Vooki incluye la aplicación Web Scanner, Rest API Scanner y la sección de informes.

3.5. Fuentes de Información

Las fuentes de información que se utilizaron en esta investigación son:

Primarias

- Pruebas Aplicadas
- Observación de Resultados obtenidos de las Pruebas

Secundarias

- Trabajos de Investigación
- Libros Especializados
- Artículos Científicos
- Tesis relacionadas con el Tema de Investigación
- Páginas de Internet con Contenido Confiable (Paginas oficiales de recursos tecnológicos)

3.6. Planteamiento de la Hipótesis

3.6.1. Hipótesis General

La aplicación de la propuesta de un método de mejores prácticas optimizará el nivel de seguridad en el desarrollo de Servicios Web RESTFul

3.6.2. Identificación de Variables

Variable Independiente: Propuesta de mejores prácticas
Variable Dependiente: Nivel de seguridad

3.6.3. Operacionalización Conceptual de Variables

Tabla 1-3: Tabla de Operacionalización Conceptual de Variables

VARIABLE	TIPO	DEFINICIÓN
Método propuesto de mejores prácticas	Independiente	Conjunto de lineamientos previamente seleccionados de un análisis de procesos existentes.
Nivel de seguridad	Dependiente	Categorización de la ausencia de riesgo de vulnerabilidad en la gestión de la información.

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

3.6.4. Operacionalización Metodológica de Variables

Tabla 2-3: Tabla de Operacionalización Metodológica de Variables

VARIABLE	INDICADOR	TÉCNICA	INSTRUMENTO/ FUENTE
Método propuesto de mejores prácticas	▪ Nivel de satisfacción en la aplicación del método.	▪ Observación	▪ Aplicativo implementado con método propuesto
Nivel de seguridad	▪ Número de vulnerabilidades detectadas en Autenticación ▪ Número de vulnerabilidades con sesión iniciada	▪ Ataques simulados a los ambientes implementados ▪ Observación de pruebas sobre los aplicativos	▪ Herramienta de monitoreo y testeo de pentesting.

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

3.7. Población y Muestra

3.7.1. Población

La población de la investigación es el conjunto de los controles especificados en el REST Security Cheat Sheet de OWASP donde detalla los parámetros a considerar en el desarrollo de Servicios Web REST seguros.

De la aplicación de dichos controles y la revisión de vulnerabilidades aplicando herramientas de pentesting se logró obtener el método de mejores prácticas de desarrollo de dichos servicios.

3.7.2. Selección de la Muestra

Para la selección de la muestra se consideró los aspectos definidos en REST Security Cheat Sheet de OWASP, tomados de forma no probabilística, partiendo desde la más transcendental en cuanto a seguridad, y que ayudaron a determinar un método óptimo en el desarrollo de estos servicios y que finalmente fueron probados al aplicar la propuesta en uno de los ambientes implementados como se define en la ***Tabla 3-3***.

Tabla 3-3: Tabla de parámetros a evaluar

No	Parámetro de seguridad
1	Access Control
2	Uso HTTPS
3	Códigos de Error
4	JWS
5	Auditoría de logs
6	CORS
7	Restricciones de Métodos
8	Validación de Tipos de Contenido y Entradas
9	Documentación

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Para generalizar los resultados de la investigación que valide la propuesta del método planteado, se aplicaron pentesting que ayudaron a validar la aplicabilidad de los nueve (9) parámetros definidos (***Tabla 3-3***).

3.8. Procedimientos Generales

Para la recolección de datos que validen la investigación se procedió a aplicar la observación con la finalidad de verificar si el método propuesto cubre todos los requerimientos de seguridad en el desarrollo de este tipo de servicios y además se revisó los resultados obtenidos de la utilización de herramientas de pentesting aplicadas a los dos escenarios del aplicativo (***Tabla 4-3***).

Tabla 4-3: Técnicas de demostración de hipótesis

VARIABLE	INDICADOR	TÉCNICA
Método Propuesto de Mejores prácticas	▪ Nivel de satisfacción en la aplicación del método.	Observación y análisis
Nivel de Seguridad	▪ Número de vulnerabilidades detectadas en Autenticación ▪ Número de vulnerabilidades con sesión iniciada	**Pruebas de Aplicativos** ▪ Utilización de herramientas Vooki, Postman para detección de vulnerabilidades

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Para la realización del presente estudio de investigación se definieron los siguientes lineamientos que otorgaron la secuencia lógica a seguir para la poder cumplir con los objetivos planteados:

1. Análisis de los métodos de desarrollo existentes definidos en OWASP
2. Levantamiento del ambiente de desarrollo necesario para la implementación de los escenarios
    a. Instalación de nodejs
    b. Instalación de Docker para gestión del contenedor de mongodb

    c. Instalación de Sublime Text, entorno de desarrollo para programar los aplicativos.
    d. Instalación de clientes postman y vooki para acceder a los servicios implementados.
3. Implementación del Escenario 1, aplicativo que contiene el modelo producto y las acciones necesarias que cumpla las características REST y desarrollado con métodos existentes.
4. Definición de mejores prácticas necesarias posterior al análisis para generar la propuesta.
5. Implementación del Escenario 2, aplicativo que contiene el modelo producto y las acciones necesarias que cumpla con las características REST aplicado la propuesta de mejores prácticas.
6. Configuración del Escenario 1
    a. Publicación del código fuente del aplicativo el repositorio de Github.
    b. Publicación del aplicativo en el servicio en la nube Heroku con sus configuraciones por defecto.
    c. Aplicación de Observación para cumplimiento de aplicación de mejores prácticas.
    d. Aplicación de pentesting utilizando la herramienta Vooki sobre la autenticación y sobre la ruta GET en sesión iniciada.
7. Configuración del Escenario 2
    a. Publicación del código fuente del aplicativo el repositorio de Github.
    b. Publicación del aplicativo en el servicio en la nube Digital Ocean en un doplet.
    c. Configuración del dominio particular y del certificado digital.
    d. Aplicación de Observación para cumplimiento de aplicación de mejores prácticas.
    e. Aplicación de pentesting utilizando la herramienta Vooki sobre la autenticación y sobre la ruta GET en sesión iniciada.
8. Observación y análisis de los resultados de resultados mediante la tabulación de toma de muestras y generación de datos estadísticos.

3.9. Instrumentos de Recolección de Datos

Para la recolección de datos para la investigación se utilizaron las herramientas Open Source explicadas anteriormente en la sección Instrumentos, como son Vooki, Postman que ayudaron a realizar la detección de vulnerabilidades en los dos escenarios.

3.10. Instrumentos para Procesar Datos Recopilados

Para el procesamiento de los datos obtenidos se utilizó software específico para tabulación y análisis estadístico como son Microsoft Excel y la herramienta de matemáticas de nivel educativo Online GeoGebra[2] para realizar el gráfico de chi-cuadrado.

3.11. Ambiente de Pruebas

Los escenarios de pruebas para la investigación, permitieron simular condiciones similares para obtener y verificar los resultados en los ambientes donde los aplicativos trabajarían comúnmente.

3.11.1. Escenarios

Para levantar el ambiente de pruebas se implementaron dos aplicativos desarrollados en nodejs usando el framework expressjs y mongodb para la gestión de los datos, lo mismos que se cumplen la siguiente arquitectura (***Figura 8-3***):

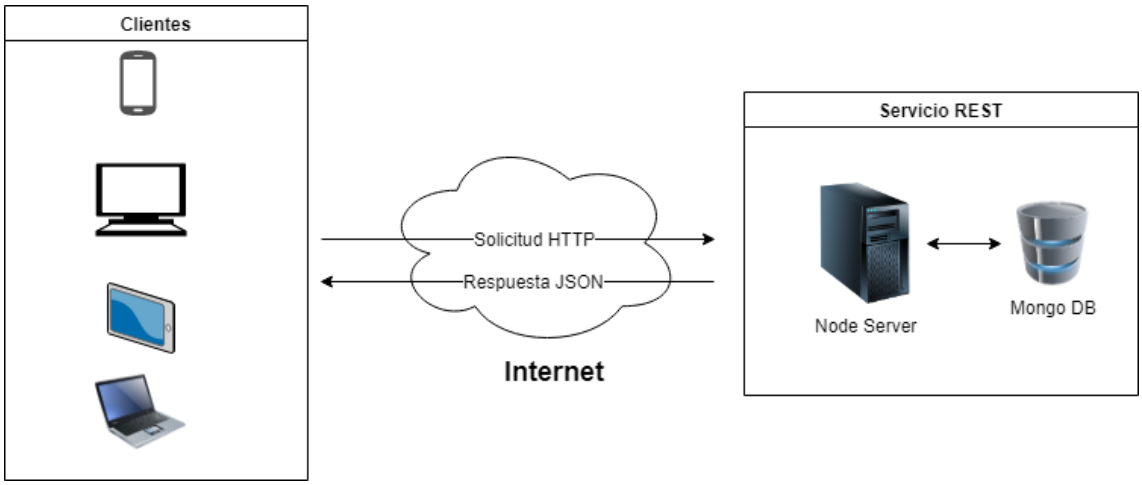


Figura 8-3: Arquitectura de Servicios Web RESTFul
Fuente: Villa Fabián, 2019

Los Servicios REST implementados en cada uno de los escenarios contempla la lógica del aplicativo enmarcado en un conjunto de acciones invisibles para el usuario final por lo que es necesario para visualizar los datos gestionados el uso de clientes que por medio de una solicitud HTTP se puede conectarse e interactuar con los servicios.

[2] Geogebra online https://www.geogebra.org/m/YQCfcR2J

Cada servicio de cada escenario fue implementado con las mismas herramientas siguiendo el estilo arquitectónico de REST con la única diferencia que al momento del desarrollo se aplicaron diferentes métodos en la codificación y el uso de la propuesta de mejores prácticas (***Figura 9-3***).

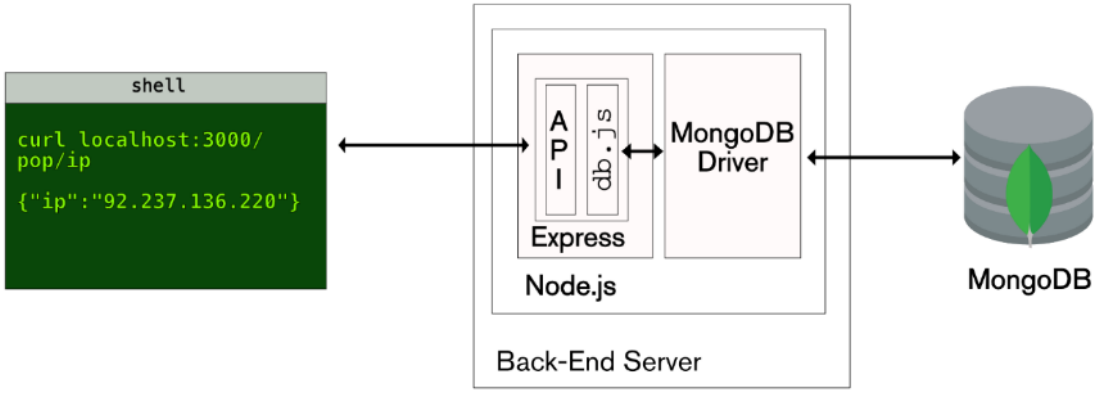


Figura 9-3: Estilo Arquitectónico de REST
Fuente: (MongoDB, Inc., 2019)

Escenario 1

El aplicativo se implementó con los métodos tradiciones de desarrollo de Servicios Web RESTFul y se publicó en un servicio de la nube llamado Heroku con la configuración estándar que éste provee.

La aplicación consta de 4 carpetas donde se codifico la funcionalidad básica de los Servicios Web RESTFul (***Figura 10-3***):

App/Model: Define el modelo del aplicativo donde consta los campos del recurso a gestionar.
Config: Define las configuraciones del aplicativo como credenciales de acceso a mongodb y credenciales de acceso al api.
Helpers: Contiene definición de funciones comunes en todo el aplicativo.
Routes: Contiene las definiciones de las rutas o métodos de los aplicativos, además aquí se codifica cada acción que se ejecuta al invocar una de estas rutas.

El archivo **app.js** es uno de los más importantes ya que en él se codifica la integración de todos los compontes definidos y codificados en las carpetas anteriores.

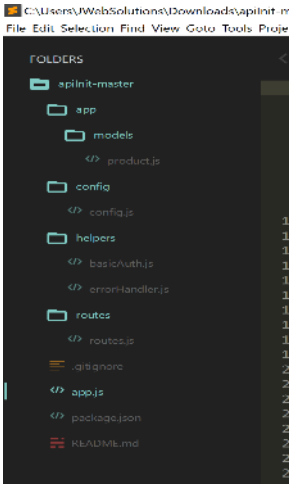


Figura 10-3*:* Estructura del código del aplicativo del Escenario 1 en Sublime Text
Fuente: Villa Fabián, 2019

En el desarrollo se usó el método de autenticación Basic que contempla el envío de las credenciales en cada solicitud hacia el servicio por parte del cliente y su implementación está en el archivo *"helpers/basicAuth.js"*.

Una vez probado en el ambiente de desarrollo se publicó una versión del servicio en el repositorio de Github para posteriormente publicarlo usando el servicio en la nube de Heroku.

En este servicio Heroku se creó una nueva aplicación *"ini-initial"*, y se enlazo con el repositorio en GitHub desde el Dashboard, esta acción permitirá que a futuro cualquier cambio que se suba a GitHub automáticamente se publique en el servicio de Heroku, además se definió las credenciales de acceso al mongodb para el almacenamiento de los datos que el aplicativo gestiona (***Figura 11-3***).

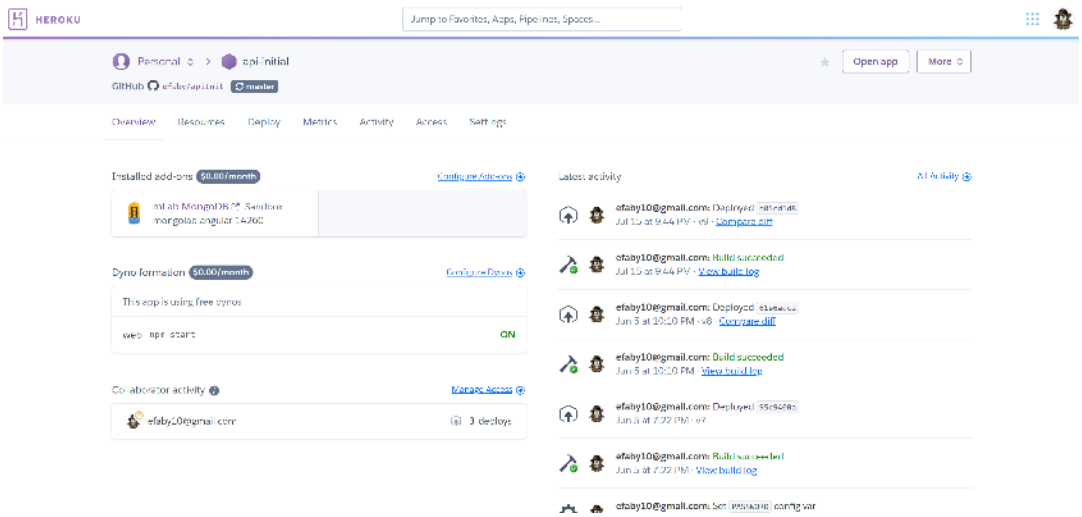


Figura 11-3*:* Dashboard de Heroku para gestión del aplicativo *"api-initial"*
Fuente: Villa Fabian, 2019

La configuración en el servicio de Heroku no permitió el uso de herramientas que permitan configurar de una manera más segura el servidor y poder contrarrestar las vulnerabilidades en este escenario como el uso de HTTP para el aplicativo RESTFul.

Figura 12-3: Visualización del aplicativo *"api-initial"*
Fuente: Villa Fabián, 2019

En la figura anterior (***Figura 12-3***) se visualiza un método público expuesto para verificar si el servicio está o no activo, recordando que los servicios son la parte invisible al usuario.

Escenario 2

El aplicativo se implementó con el uso del método de mejores prácticas propuesto de desarrollo de Servicios Web RESTFul y se publicó en un droplet del servicio en la nube Digital Ocean para la configuración del software necesario ya que el servicio anterior no facilitaba mucho la configuración optima de seguridad para desplegar la aplicación, así como también la instalación del certificado digital.

La aplicación consta de 5 carpetas donde se codifico la funcionalidad en base al método propuesto para el desarrollo de Servicios Web RESTFul (***Figura 13-3***):

App/Model: Define el modelo del aplicativo donde consta los campos del recurso a gestionar.
App/Controller: Define las acciones del aplicativo que se ejecutarán al momento de invocar un recurso.
Config: Define las configuraciones del aplicativo como credenciales de acceso a mongodb y credenciales de acceso al api y secret tokens para la autenticación.
Services: Contiene la definición de la funcionalidad de validación de tokens en el proceso de autenticación.
Routes: Contiene las definiciones de las rutas o métodos de los aplicativos, y se re direcciona a las acciones definidas en los controllers además contiene la notación para la documentación.
Test: Contiene los test unitarios para la verificación del correcto funcionamiento del aplicativo.

De igual manera el archivo **app.js** es uno de los más importantes ya que en él se codifica la integración de todos los compontes definidos y codificados en las carpetas anteriores y además se incluye las librerías de seguridad necesarias para una óptima configuración de cabeceras y del aplicativo en sí.

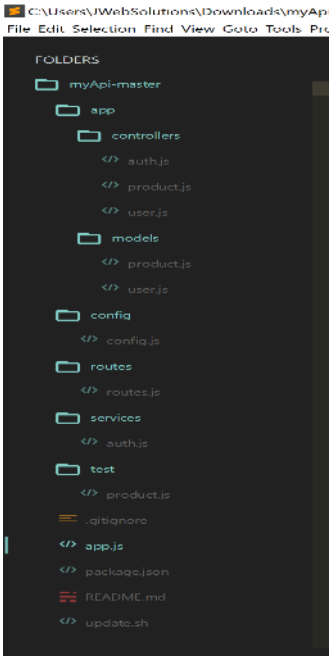


Figura 13-3*:* Estructura del código del aplicativo del Escenario 2 en Sublime Text
Fuente: Villa Fabián, 2019

En el desarrollo se usó el método de autenticación por tokens usando JWT que contempla la implementación de un método de autenticación donde se le envía las credenciales y devuelve un token de acceso necesario para el consumo de los recursos del servicio en cada solicitud por parte del cliente.

Una vez probado en el ambiente de desarrollo ejecutando las pruebas unitarias se publicó una versión del servicio en el repositorio de Github para posteriormente poder publicarlo desde el droplet mediante la ejecución de un script.

En el dropet se instaló las herramientas necesarias para que el aplicativo funcione, así como también se instaló el certificado digital y se re direccionó un subdominio propio para darle mayor relevancia y poder obtener resultados confiables con elementos propios de un ambiente de producción.

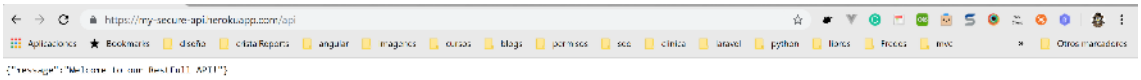


Figura 14-3*:* Visualización del aplicativo *"my-api"*
Fuente: Villa Fabián, 2019

En la figura anterior (***Figura 14-3***) se visualiza un método público expuesto para verificar si el servicio está o no activo, recordando que los servicios son la parte invisible al usuario.

3.11.2. Resultados

Para obtener los resultados necesarios para el estudio, se definieron las siguientes pruebas para cado uno de los escenarios:

Una vez implementado y publicados los Escenarios, se procedió a realizar la verificación del cumplimiento de aplicabilidad en cada uno de los parámetros definidos en la ***Tabla 3-3*** usando la herramienta POSTMAN, ejecutando en éste cliente las acciones principales de HTTP (POST, PUT, GET, DELETE) y en base a las respuestas que se obtiene con la herramienta se registraron los datos en una tabla comparativa de los dos Escenarios.

Para la obtención de los datos para verificar el nivel de seguridad se utilizó la herramienta VOOKI que permitió realizar el pentesting y obtener las vulnerabilidades en cada uno de los Escenarios y sobre dos acciones como son en el proceso de Autenticación y en la solicitud GET en sesión iniciada.

Posterior a la aplicación de las pruebas definidas para los dos escenarios, se recolectaron los datos numéricos necesarios para aplicar la distribución, con el objetivo de demostrar que el método propuesto permite crear Servicios Web RESTFul seguros.

CAPÍTULO IV

4. RESULTADOS Y DISCUSIÓN

4.1. Presentación de Resultados

En la siguiente sección se presenta el análisis de los resultados obtenidos en la investigación aplicando los métodos y técnicas definidos, así como su relación con los objetivos y la hipótesis planteada.

En base a dichos resultados, se puede observar la diferencia existente entre los aplicativos implementados en los escenarios, y se obtiene la conclusión de que el método propuesto es el indicado para obtener un Servicio Web RESTFul seguro.

4.2. Procesamiento y Análisis

En la presente investigación se aplicó la observación y un chequeo de cumplimiento de los parámetros a evaluar para la demostración de la variable independiente y el uso de las herramientas pentesting para la demostración de la variable dependiente aplicando las pruebas sobre los dos escenarios planteados.

4.3. Valoración de la Variable Independiente

4.3.1. Variable Independiente: Método propuesto de mejores prácticas

Para su valoración se procedió a realizar una observación y así validar el cumplimiento del uso los parámetros planteados para la evaluación en cada escenario.

4.3.2. Indicador Nivel de Satisfacción

Nivel de satisfacción en la aplicación del método: Para la medición de este indicador se utilizó la escala de Likert (Netquest, 2014) que permitió tener una valoración del nivel de satisfacción de la aplicación del método (***Tabla 1-4***).

Tabla 1-4: Escala Likert

Escala	Siempre	Casi siempre	Algunas veces	Muy pocas veces	Nunca
Valoración	5	4	3	2	1

Fuente: (Netquest, 2014)
Realizado por: Villa Fabián, 2019

La observación realizada sobre los escenarios definidos utilizando la herramienta POSTMAN en base a los resultados obtenidos de ésta y aplicando la escala anterior (***Tabla 1-4***), se define la siguiente ***Tabla 2-4***.

Tabla 2-4: Parámetros evaluados en los escenarios definidos

No	Parámetro	Escenario 1	Escenario 2
1	Uso de certificado digital	5	5
2	Control de acceso a sus recursos	2	4
3	Uso de JWT	1	5
4	Restricción de Métodos HTTP no disponibles	1	5
5	Uso de CORS	1	4
6	Presentación correcta de códigos de error	2	4
7	Validación de tipos de contenidos en sus entradas	2	4
8	Auditoria de registro de logs de acceso y error	2	4
9	Documentación	1	5
	Total	**17**	**40**

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Análisis e Interpretación de Resultados:

De estos datos el porcentaje de aplicabilidad se calcula como valor de 45 al 100% ya que indicaría que siempre se aplica los nueve (9) parámetros planteados a evaluar. Dicho esto, los valores de aplicabilidad obtenidos en la observación corresponden al 37,78% en el Escenario 1 y el 88.89% en el Escenario 2 al aplicar el método propuesto de lo que se puede apreciar una mejora en el desarrollo seguro de estos servicios.

4.4. Valoración de la Variable Dependiente

4.4.1. Variable Dependiente: Nivel de Seguridad

Para su valoración se utilizó la herramienta de pentesting Vooki aplicada a los dos escenarios implementados.

4.4.2. Indicador Vulnerabilidades en Autenticación

Número de vulnerabilidades detectadas en Autenticación: Para la medición de este indicador se utilizó la herramienta Vooki aplicando el scanner de vulnerabilidad sobre la url de Autenticación en el Método POST.

Identificación de vulnerabilidades sobre el Escenario 1, (*Figura 1-4*)

Servicio Restful initApi
Url: https://api-initial.herokuapp.com/api/authenticate

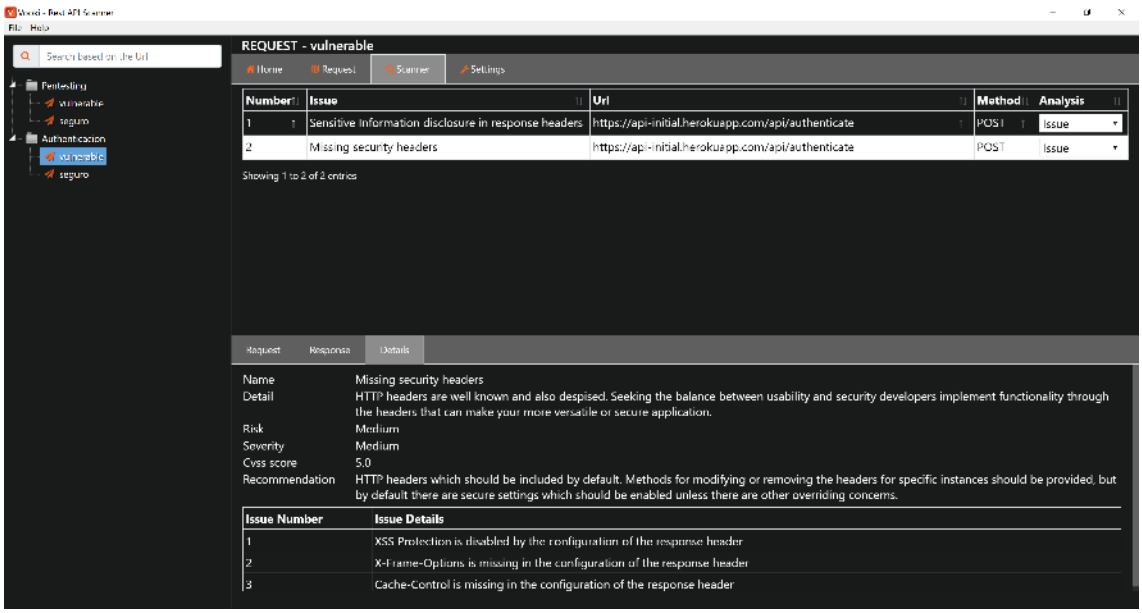


Figura 1-4: Captura de pentesting en Autenticación Vooki en Escenario 1
Realizado por: Villa Fabián, 2019

Tabla 3-4: Vulnerabilidades en Autenticación encontradas en Escenario 1

Vulnerabilidad	Nivel	Número	Descripción
Divulgación de información sensible en encabezados de respuesta	Medio	2	Presencia de información sensible de configuración el ambiente implementado
Ausencia de encabezados de seguridad	Medio	3	Presencia de información de configuración por defecto
Ausencia de endpoint de autheticacion	Bajo	1	Basic Authentication, se requiere autenticación en cada request

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Identificación de Vulnerabilidades sobre el Escenario 2, (***Figura 2-4***)

Servicio Restful Myapi

Url: https://myapi.jwebsolutions.com.ec/api/authenticate

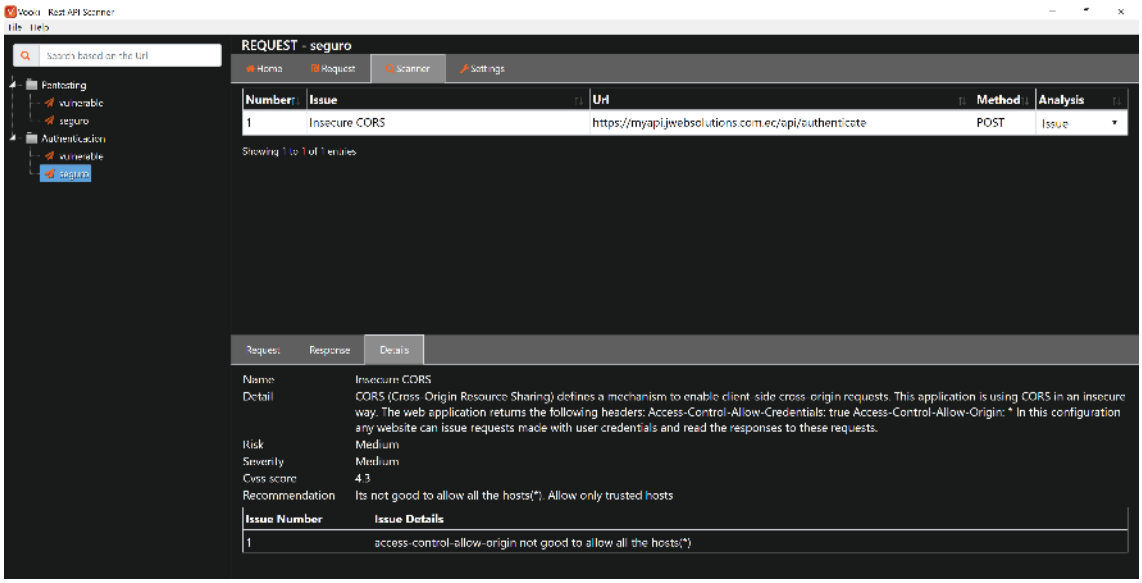


Figura 2-4: Captura de pentesting en Autenticación Vooki en Escenario 2
Realizado por: Villa Fabián, 2019

Por ser una vulnerabilidad esperada de manera intencional se la califica con un nivel "Bajo" (***Tabla 4-4***).

Tabla 4-4: Vulnerabilidades en Autenticación encontradas en Escenario 2

Vulnerabilidad	Nivel	Número	Descripción
CORS Inseguros	Bajo	1	Para la ejecución del pentensting se habilito el acceso a todas las urls con la opción de "access-control-allow-origin:*" para dar acceso a la herramienta

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

4.4.3. Indicador Vulnerabilidades en Sesión Iniciada

Número de Vulnerabilidades con Sesión Iniciada: Para la medición de este indicador se utilizó la herramienta Vooki aplicando el scanner de vulnerabilidad sobre la url de Listado de Productos en el Método GET con un usuario autenticado.

Identificación de Vulnerabilidades sobre el Escenario 1, (***Figura 3-4***)

Servicio Restful initApi

Url: https://api-initial.herokuapp.com/api/products/

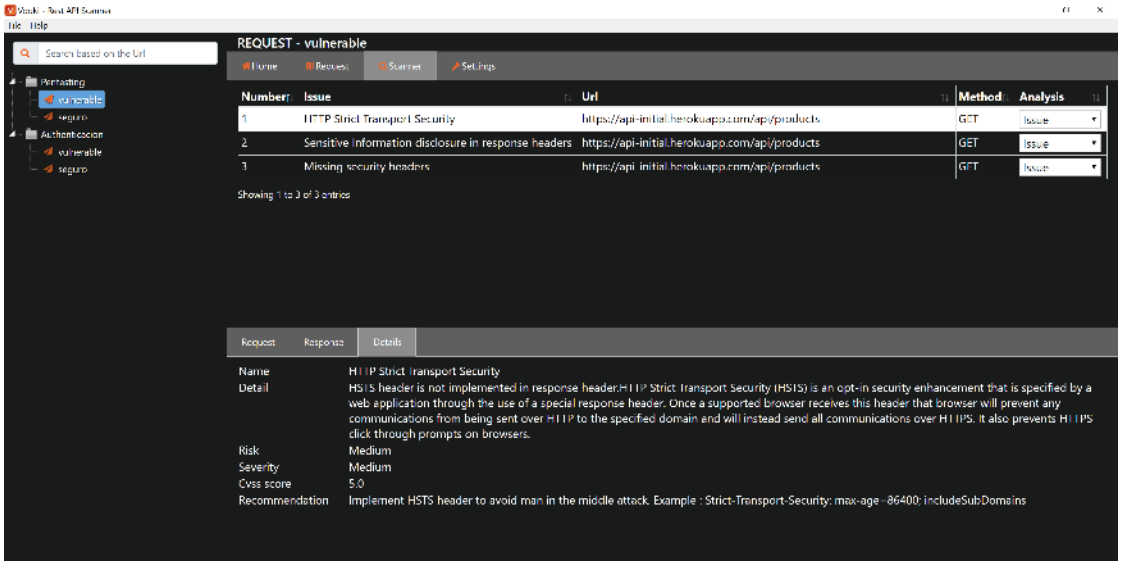


Figura 3-4: Captura de pentesting en Sesión Iniciada Vooki en Escenario 1
Realizado por: Villa Fabián, 2019

Tabla 5-4: Vulnerabilidades en Sesión Iniciada encontradas en Escenario 1

Vulnerabilidad	Nivel	Número	Descripción
Seguridad de Transporte HTTP	Medio	1	Vulnerabilidad que permite ataque de Hombre en la mitad ya que si contenido no está cifrado
Divulgación de Información sensible en Encabezados de Respuesta	Medio	2	Presencia de información sensible de configuración el ambiente implementado
Ausencia de Encabezados de Seguridad	Medio	4	Presencia de información de configuración por defecto
Presencia de Credenciales de Acceso en cada Request	Bajo	1	Presencia de información sensible del usuario
Manejo de Errores no implementado	Alto	1	Al ejecutar el pentesting el aplicativo dejo de responder

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Identificación de Vulnerabilidades sobre el Escenario 2, (***Figura 4-4***)

Servicio Restful Myapi

Url: https://myapi.jwebsolutions.com.ec/api/product

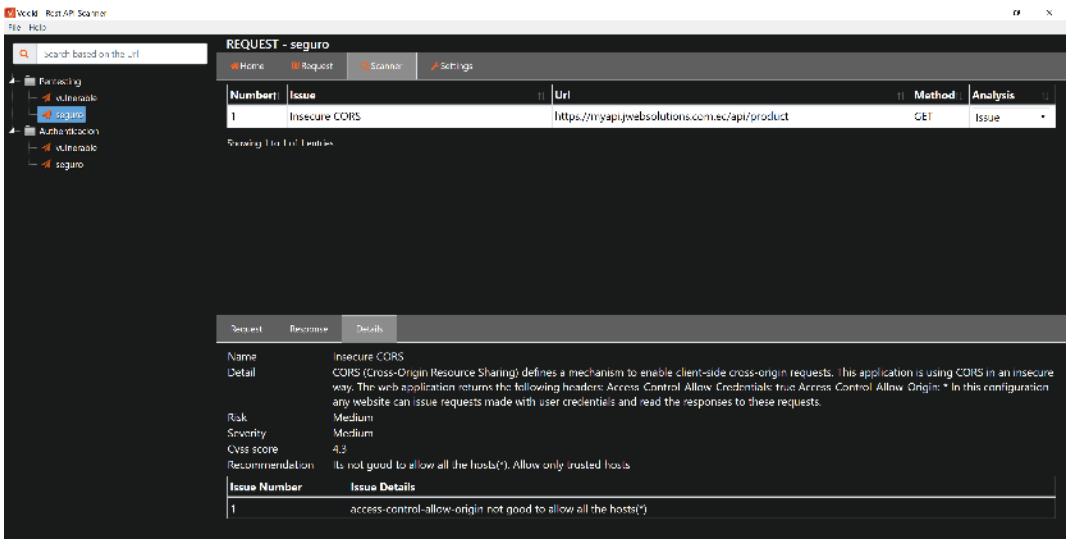


Figura 4-4: Captura de pentesting en Sesión Iniciada Vooki en Escenario 2
Realizado por: Villa Fabián, 2019

Por ser una vulnerabilidad esperada de manera intencional se la califica con un nivel "Bajo" (***Tabla 6-4***).

Tabla 6-4: Vulnerabilidades en Sesión Iniciada encontradas en Escenario 2

Vulnerabilidad	Nivel	Número	Descripción
CORS Inseguros	Bajo	1	Para la ejecución del pentensting se habilito el acceso a todas las urls con la opción de "access-control-allow-origin:*" para dar acceso a la herramienta

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Una vez aplicado los pentesting sobre los dos escenarios y una vez obtenido los resultados de las vulnerabilidades se obtiene la siguiente ***Tabla 7-4*** con el total de ellos entre los dos indicadores seleccionando las vulnerabilidades sin repetición:

Tabla 7-4: Resumen de resultados obtenidos en los pentesting

Vulnerabilidades encontradas	Frecuencia		Porcentaje de Reducción
	Escenario 1	Escenario 2	
Altas	1	0	100%
Medias	7	0	100%
Bajas	2	1	50%
Total	**10**	**1**	**90%**

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

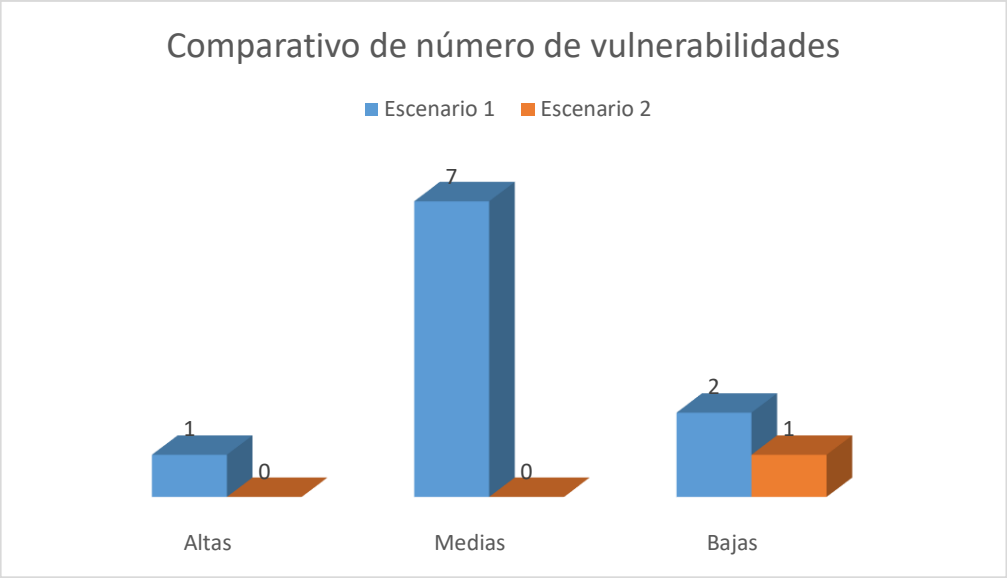


Gráfico 1-4: Comparativo número de vulnerabilidades
Realizado por: Villa Fabián, 2019

Análisis e Interpretación de Resultados:

Para realizar una adecuada medición de la reducción de vulnerabilidades existentes en un Servicio Web RESTFul y aplicando el método sugerido podemos denotar que se ha reducido en un 100% las vulnerabilidades de Nivel Alto y Nivel Medio encontradas, así como también un 50% de las vulnerabilidades bajas dando como resultado total un 90% de reducción con la aplicación del método propuesto.

4.5. Comprobación Estadística De La Hipótesis

Para la comprobación de la hipótesis general "La aplicación de la propuesta de un método de mejores prácticas optimizará el nivel de seguridad en el desarrollo de Servicios Web RESTFul", se utilizó estadística referencial aplicando la prueba **Chi-Cuadrado(X^2)**.

Posterior a la realización de los diferentes análisis, y con los datos obtenidos se procede a definir la hipótesis de investigación *Hi* y la Hipótesis nula *Ho* a ser consideradas:
Hi: "La aplicación de la propuesta de un método de mejores prácticas **si** optimizará el nivel de seguridad en el desarrollo de Servicios Web RESTFul"
Ho: "La aplicación de la propuesta de un método de mejores prácticas **no** optimizará el nivel de seguridad en el desarrollo de Servicios Web RESTFul"

La siguiente ***Tabla 8-4*** contiene la información de las frecuencias de valores encontrados, los mismos que serán utilizados para el cálculo en la prueba.

Tabla 8-4: Frecuencias de Valores Encontrados

	Escenario 1	Escenario 2	Total
Vulnerabilidades Encontradas	10	1	11
Vulnerabilidades Solventadas	0	9	9
Total	10	10	20

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

Para la obtención de la tabla de frecuencias esperadas se aplica la siguiente fórmula en cada valor de la tabla.

$$\text{Fe}= \frac{\text{total columna} * \text{total fila}}{\text{suma total}}$$

Tras la aplicación de la fórmula en cada valor de la tabla anterior obtendremos la siguiente ***Tabla 9-4*** de frecuencias esperadas.

Tabla 9-4: Frecuencias Esperadas

	Escenario 1	Escenario 2	Total
Vulnerabilidades Encontradas	5.5	5.5	11
Vulnerabilidades Solventadas	4.5	4.5	9
Total	10	10	20

Fuente: Villa Fabián, 2019
Realizado por: Villa Fabián, 2019

A continuación, se calcula el valor de X2 mediante la siguiente fórmula

$$x^2 = \sum \frac{(FO - FE)^2}{FE}$$

Donde:

FO: Frecuencia Observada por celda

FE: Frecuencia Esperada por celda

$$x^2 = \frac{(10 - 5.5)^2}{5.5} + \frac{(0 - 4.5)^2}{4.5} + \frac{(1 - 5.5)^2}{5.5} + \frac{(9 - 4.5)^2}{4.5}$$

$$x^2 = 3.68 + 4.5 + 3.68 + 4.5$$

$$x^2 = 16.36$$

El siguiente paso a seguir es el cálculo de los grados de libertad

$$v = \ (r-1) \times (k-1)$$

Donde:

r: número de filas

k: número de Columnas

$$v = (2-1) \ \times (2-1)$$

$$v = 1$$

En base a la tabla de la distribución de Chi-Cuadrado (***Figura 5-4***), y determinando el valor de significancia de 0.05% obtenemos el punto crítico con 1 como valor de grados de libertad.

TABLA 3-Distribución Chi Cuadrado χ^2

P = Probabilidad de encontrar un valor mayor o igual que el chi cuadrado tabulado, ν = Grados de Libertad

ν/p	0,001	0,0025	0,005	0,01	0,025	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5
1	10,8274	9,1404	7,8794	6,6349	5,0239	3,8415	2,7055	2,0722	1,6424	1,3233	1,0742	0,8735	0,7083	0,5707	0,4549
2	13,8150	11,9827	10,5965	9,2104	7,3778	5,9915	4,6052	3,7942	3,2189	2,7726	2,4079	2,0996	1,8326	1,5970	1,3863
3	16,2660	14,3202	12,8381	11,3449	9,3484	7,8147	6,2514	5,3170	4,6416	4,1083	3,6649	3,2831	2,9462	2,6430	2,3660
4	18,4662	16,4238	14,8602	13,2767	11,1433	9,4877	7,7794	6,7449	5,9886	5,3853	4,8784	4,4377	4,0446	3,6871	3,3567
5	20,5147	18,3854	16,7496	15,0863	12,8325	11,0705	9,2363	8,1152	7,2893	6,6257	6,0644	5,5731	5,1319	4,7278	4,3515
6	22,4575	20,2491	18,5475	16,8119	14,4494	12,5916	10,6446	9,4461	8,5581	7,8408	7,2311	6,6948	6,2108	5,7652	5,3481
7	24,3213	22,0402	20,2777	18,4753	16,0128	14,0671	12,0170	10,7479	9,8032	9,0371	8,3834	7,8061	7,2832	6,8000	6,3458
8	26,1239	23,7742	21,9549	20,0902	17,5345	15,5073	13,3616	12,0271	11,0301	10,2189	9,5245	8,9094	8,3505	7,8325	7,3441
9	27,8767	25,4625	23,5893	21,6660	19,0228	16,9190	14,6837	13,2880	12,2421	11,3887	10,6564	10,0060	9,4136	8,8632	8,3428
10	29,5879	27,1119	25,1881	23,2093	20,4832	18,3070	15,9872	14,5339	13,4420	12,5489	11,7807	11,0971	10,4732	9,8922	9,3418
11	31,2635	28,7291	26,7569	24,7250	21,9200	19,6752	17,2750	15,7671	14,6314	13,7007	12,8987	12,1836	11,5298	10,9199	10,3410
12	32,9092	30,3182	28,2997	26,2170	23,3367	21,0261	18,5493	16,9893	15,8120	14,8454	14,0111	13,2661	12,5838	11,9463	11,3403
13	34,5274	31,8830	29,8193	27,6882	24,7356	22,3620	19,8119	18,2020	16,9848	15,9839	15,1187	14,3451	13,6356	12,9717	12,3398
14	36,1239	33,4262	31,3194	29,1412	26,1189	23,6848	21,0641	19,4062	18,1508	17,1169	16,2221	15,4209	14,6853	13,9961	13,3393
15	37,6978	34,9494	32,8015	30,5780	27,4884	24,9958	22,3071	20,6030	19,3107	18,2451	17,3217	16,4940	15,7332	15,0197	14,3389
16	39,2518	36,4555	34,2671	31,9999	28,8453	26,2962	23,5418	21,7931	20,4651	19,3689	18,4179	17,5646	16,7795	16,0425	15,3385
17	40,7911	37,9462	35,7184	33,4087	30,1910	27,5871	24,7690	22,9770	21,6146	20,4887	19,5110	18,6330	17,8244	17,0646	16,3382
18	42,3119	39,4220	37,1564	34,8052	31,5264	28,8693	25,9894	24,1555	22,7595	21,6049	20,6014	19,6993	18,8679	18,0860	17,3379
19	43,8194	40,8847	38,5821	36,1908	32,8523	30,1435	27,2036	25,3289	23,9004	22,7178	21,6891	20,7638	19,9102	19,1069	18,3376
20	45,3142	42,3358	39,9969	37,5663	34,1696	31,4104	28,4120	26,4976	25,0375	23,8277	22,7745	21,8265	20,9514	20,1272	19,3374
21	46,7963	43,7749	41,4009	38,9322	35,4789	32,6706	29,6151	27,6620	26,1711	24,9348	23,8578	22,8876	21,9915	21,1470	20,3372
22	48,2676	45,2041	42,7957	40,2894	36,7807	33,9245	30,8133	28,8224	27,3015	26,0393	24,9390	23,9473	23,0307	22,1663	21,3370
23	49,7276	46,6231	44,1814	41,6383	38,0756	35,1725	32,0069	29,9792	28,4288	27,1413	26,0184	25,0055	24,0689	23,1852	22,3369
24	51,1790	48,0336	45,5584	42,9798	39,3641	36,4150	33,1962	31,1325	29,5533	28,2412	27,0960	26,0625	25,1064	24,2037	23,3367
25	52,6187	49,4351	46,9280	44,3140	40,6465	37,6525	34,3816	32,2825	30,6752	29,3388	28,1719	27,1183	26,1430	25,2218	24,3366
26	54,0511	50,8291	48,2898	45,6416	41,9231	38,8851	35,5632	33,4295	31,7946	30,4346	29,2463	28,1730	27,1789	26,2395	25,3365
27	55,4751	52,2152	49,6450	46,9628	43,1945	40,1133	36,7412	34,5736	32,9117	31,5284	30,3193	29,2266	28,2141	27,2569	26,3363

Figura 5-4: Tabla de Distribución Chi Cuadrado
Fuente: (Universidad Carlos III de Madrid - Departamento de Estadistica, 2019)

$$x^2 \ crítico = 3{,}8415$$

Dado los datos anteriores Ho debe ser aceptada si sucede el siguiente condicionante

$$x^2 \ Calculado \ \leq \ x^2 crítico$$

Caso contrario se rechaza Ho y se Acepta Hi.

Con los datos obtenidos anteriormente donde X2 = 16.36 y X2 critico = 3.8415 se puede aplicar el criterio de decisión y obtenemos que (***Gráfico 2-4***):

$$x^2\ 16.36\ >\ x^2\ crítico\ 3.8415$$

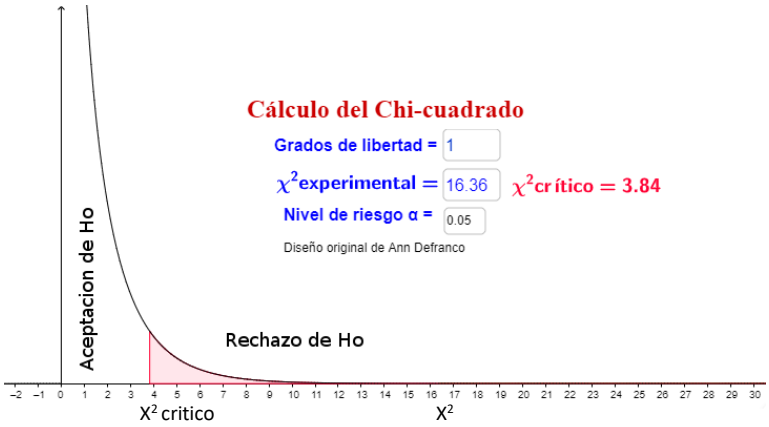


Gráfico 2-4: Chi-Cuadrado y Criterios de Aceptación de Ho
Fuente: (GeoGebra, 2019)
Realizado por: Villa Fabián, 2019

Interpretación y Análisis

En consecuencia, con los datos obtenidos y como se aprecia en la gráfica se concluye que se rechaza la Hipótesis nula Ho y se acepta la Hipótesis alternativa Hi con un nivel de confianza del 95% y un nivel de significancia de 5%.

CAPÍTULO V

5. PROPUESTA

5.1. Determinación de la Propuesta

En el presente capítulo se presenta una propuesta de un modelo de mejores prácticas para el desarrollo de Servicios Web RESTFul seguros en donde se trata de abarcar todos y cada uno de los puntos claves de vulnerabilidades presentadas en este tipo de arquitecturas de software.

Para esta propuesta, se tomó como base los aspectos definidos en REST Security Cheat Sheet de OWASP, en él se exponen los principales lineamientos a seguir para la implementación de Servicios Web RESTFul seguros, considerando que OWASP es una comunidad abierta dedicada a habilitar organizaciones para desarrollar comprar y mantener aplicaciones confiables sin ánimo de lucro.

También se considerarán para esta propuesta los lineamientos revisados en el marco teórico de este documento, no tomados en cuenta por OWASP, pero también considerados importantes como patrón de implementación, versionamiento y documentación.

La presente propuesta establece un método de mejores prácticas de desarrollo de Servicios Web Restful y pretende ser aplicable para la mayoría de los lenguajes web actualmente vigentes.

5.2. Propuesta de un Método de Mejores Prácticas para el Desarrollo de Servicios Web RESTFul

Para un óptimo desarrollo de Servicios Web RESTFul es necesario que el servicio a implementar tenga las siguientes consideraciones, cabe mencionar que los ítems a mencionar forman una guía de mejores prácticas para el desarrollo e implementación (***Figura 1-5***).

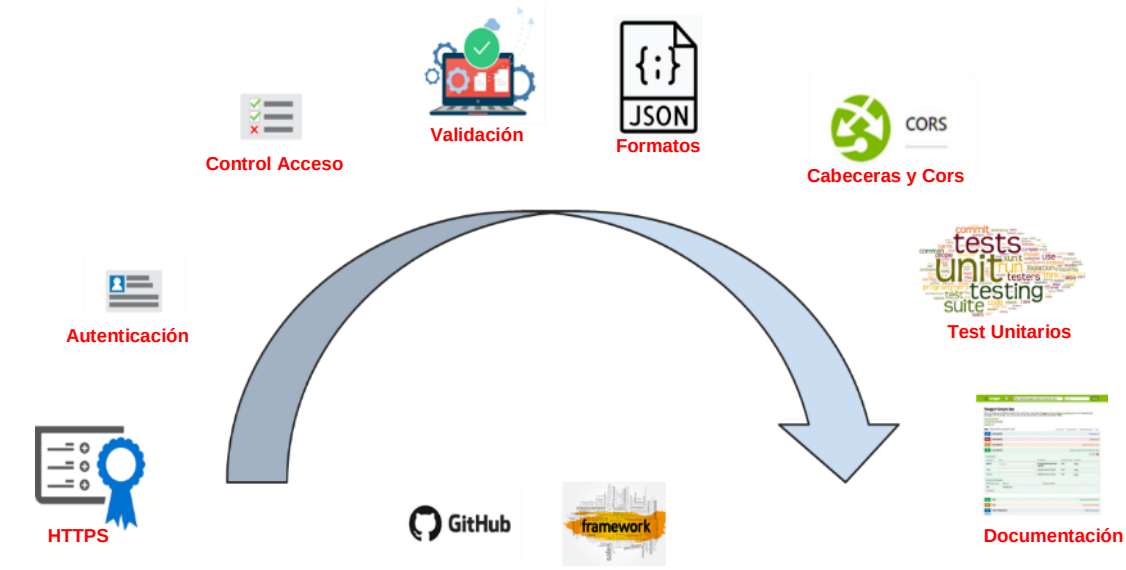


Figura 1-5: Flujo de guía de mejores prácticas propuesto
Realizado por: Villa Fabián, 2019

A continuación, se detalla cada uno de los ítems mencionados en la figura anterior:

5.2.1. Uso de Certificados Digitales

El uso de un protocolo cifrado como lo es HTTPS con cifrado SSL permite la trasportación de los datos mediante una conexión segura evitando la interceptación de la comunicación, autenticar el servicio y garantizar la integridad de los datos transmitidos.

Para el uso de este ítem se puede utilizar certificados por una Autoridad Certificadora y que por lo general lo vende el proveedor de servicios de internet ISP, y son de tipo:

Dominio Único: Protegen un único dominio
Multidominio: Protegen múltiples dominios
Wildcard: Protegen ilimitadamente los subdominios de un solo dominio.

En la presente investigación se usó un tipo especial de Certificado Digital gratuito denominado Let´s Encrypt y se lo implementó para tener el escenario seguro (***Figura 2-5***).

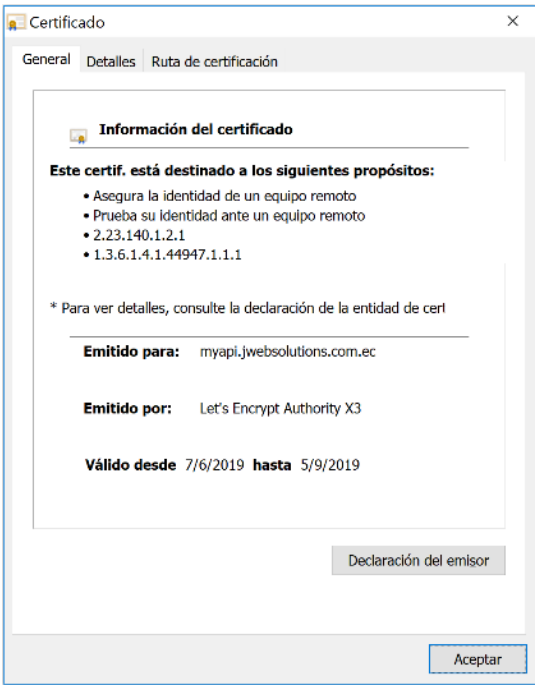


Figura 2-5: Datos de Certificado Digital Instalado
Realizado por: Villa Fabián, 2019

5.2.2. Autenticación con Métodos Estándar

El proceso de identificación de un cliente hacia el servidor se denomina autenticación que por medio de credenciales (usuario y contraseña), se valida si puede o no accederá los recursos. Por tal virtud en la implementación de un Servicio Web RESTFul la implementación por medio de métodos estándar como OAuth versión 2 o Json Web Token JWT son recomendables ya que gestionan tokens en cada petición evitando así la exposición de los datos sensibles del usuario como son sus credenciales.

En la presente investigación se usó JWT como método de autenticación (***Figura 3-5***).

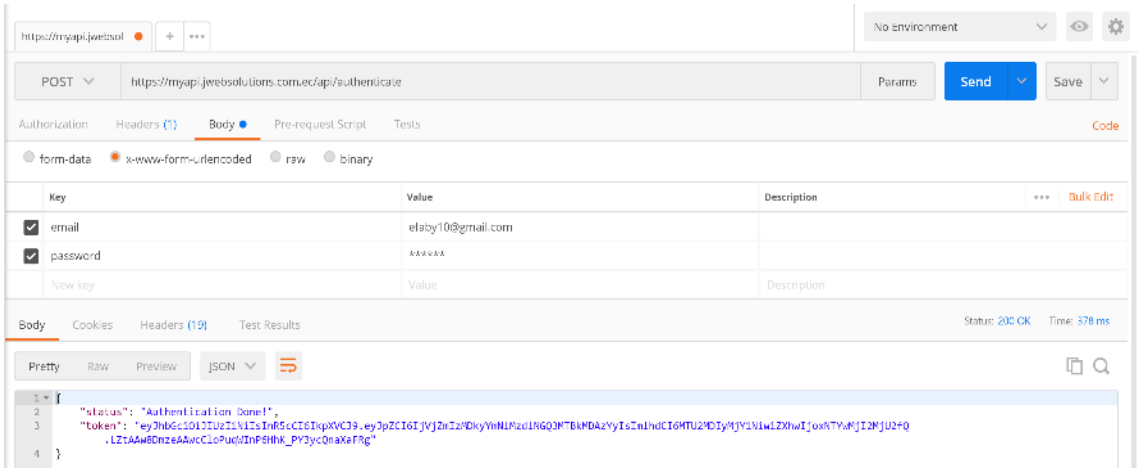


Figura 3-5: Resultado de la autenticación en el aplicativo
Realizado por: Villa Fabián, 2019

5.2.3. Control de Acceso y Restricción de Métodos HTTP

El control de acceso a los métodos definidos y no definidos es uno de los más importantes ítems a cumplir en la implementación de Servicios Web RESTFul porque pueden convertirse en brechas de seguridad comprometiendo la disponibilidad del servicio ya que se convertirían en puntos vulnerables para provocar una denegación de servicio si no están debidamente restringidos.

En la presente investigación se implementó la respuesta del código 401 a todos los urls que se intente acceder sin su debido Access Token o a su vez emitiendo un error 404 para cuando la url no exista (***Figura 4-5***).

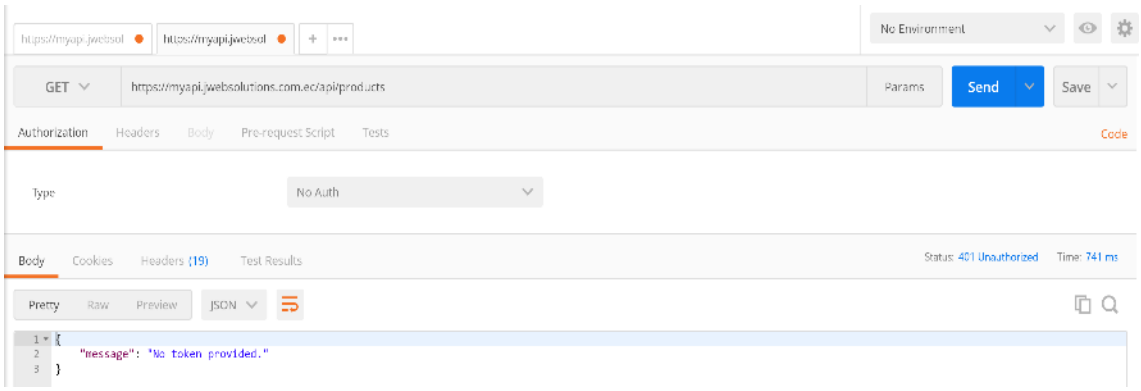


Figura 4-5: Mensaje de error con código 401
Realizado por: Villa Fabián, 2019

5.2.4. Validación de Contenido

Si bien es cierto que la validación de los datos se debería en su mayor parte, realizarse en el lado del Cliente también es necesario la validación de los datos en el servidor y es por ello que

necesariamente la validación del contenido es importante ya que también el servidor se debe proteger de datos mal intencionados.

En la presente investigación se implementó la respuesta del código 500 para indicar que hubo datos mal proporcionados (***Figura 5-5***).

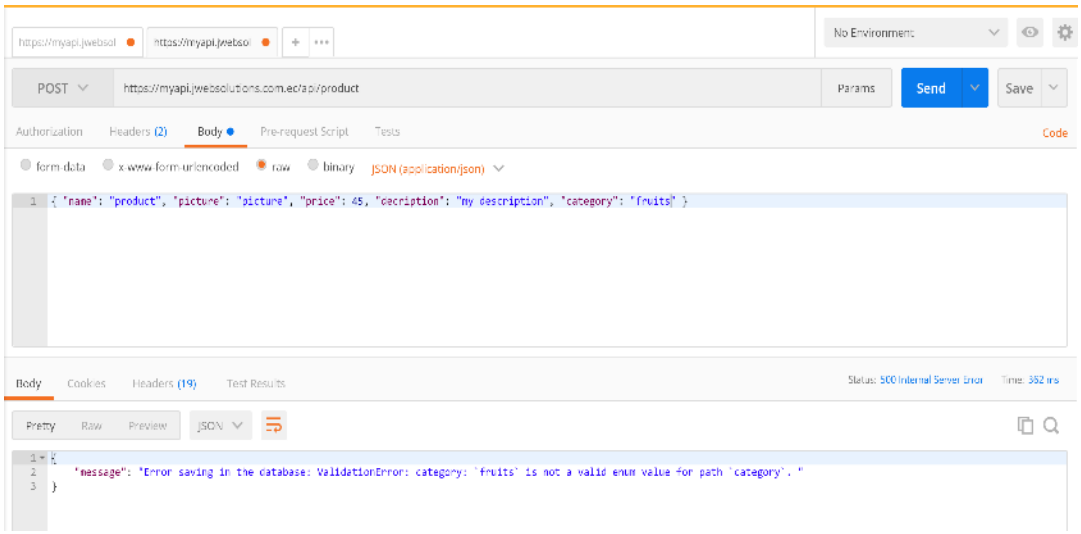


Figura 5-5: Validación del campo "category"
Realizado por: Villa Fabián, 2019

5.2.5. Uso de Formatos de Datos Estándares

En la implementación de Servicios Web REST el uso de formatos de dato estándares también es un punto importante ya que de ello se deriva la usabilidad del aplicativo. Existen diferentes tipos de datos que se utilizan en RESTFul entre ellos tenemos Json, xml o csv.

En la presente investigación se utilizó JSON (application/json) como formato de datos a gestionar en las entradas y salidas de datos (***Figura 6-5***).

5.2.6. Configuración de Cabeceras Seguras y CORS

La configuración de Cabeceras Seguras permite una comunicación optima sin la exposición de información sensible o la ausencia de configuración de encabezados seguros que eviten ataques como el XSS o el clickjacking en X-Frame-Options.

CORS son mecanismos que se utiliza en las cabeceras y permite restringir el acceso a nuestro servicio desde el origen de la petición. Este mecanismo ayuda a definir que o cuales son los

orígenes que vamos a dar acceso a nuestro servicio por medio de la cabecera “access-contro-allow-origin”.

En la presente investigación se configuro la cabecera con “*” para permitir que se puedan ejecutar los pentesting desde las herramientas que no tenían un dominio definido y se configuro las cabeceras X-Frame-Options, X-Content-Type-Options y Cache-Control.

5.2.7. Test Unitarios

En el mundo del desarrollo de software se han implementado mecanismos que ayudan a comprobar que el aplicativo que se esté desarrollando no tenga errores y esté operativo para su despliegue a producción. Es por ello que se han desarrollado conjuntos de pruebas sobre lo implementado que se denominan Test Unitarios o Unit Test que son funciones que están diseñados para comprobar que el código principal está funcionando como se esperaba.

En la presente investigación se implementó los test unitarios utilizando la librería para nodejs “mocha” que al ejecutarse muestra la siguiente salida (***Figura 6-5***).

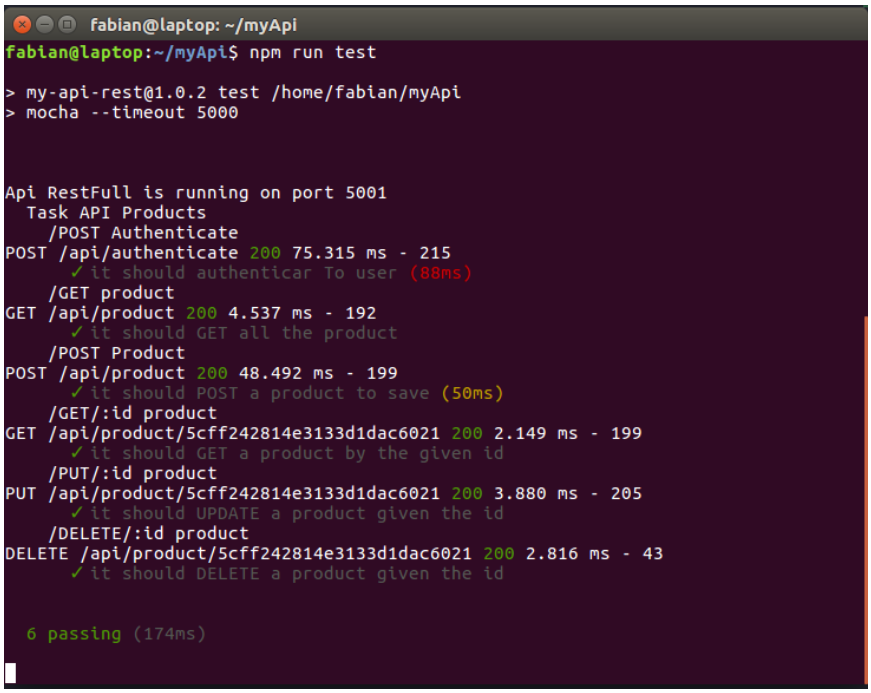


Figura 6-5: Ejecución de la pruebas unitarias en el ambiente de desarrollo
Realizado por: Villa Fabián, 2019

Los test unitarios siempre se los ejecuta en el equipo del desarrollador nunca en ambientes de producción.

5.2.8. Documentación

La aceptación y el óptimo rendimiento de un aplicativo se logra cuando este se ejecute en sus máximas capacidades y solo se pude lograr cuando se lo utiliza adecuadamente con el uso de su documentación. Este sin duda es uno de los ítems más importantes y menos implementado en el mundo del desarrollo del software ya que por lo general los aplicativos se los crea intuitivos dando una falsa sensación de no requerir una documentación que oriente su uso.

En los Servicios Web RESTFul la documentación se centra en publicar los endpoints disponibles, los parámetros de entrada y los formatos de salida de información. Así como también el método de autenticación para hacer uso de ellos.

En la presente investigación se usó el framework de documentación llamado Swagger que permite una vez creado el Servicio Web RestFul definir y generar una página donde se publican los endpoints disponibles en el servicio (***Figura 7-5***).

url: https://myapi.jwebsolutions.com.ec/api-docs/

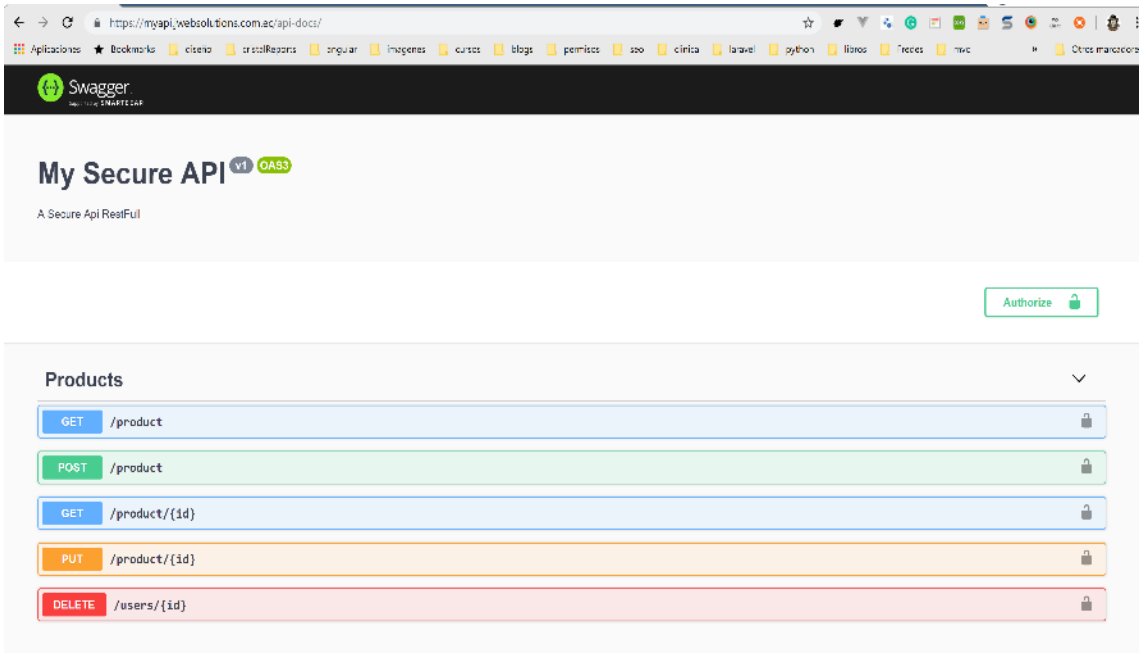


Figura 7-5: Página de documentación del aplicativo
Realizado por: Villa Fabián, 2019

5.3. Consideraciones Adicionales

Además de los ítems mencionados anteriormente también se puede sugerir para un mejor desarrollo y mantenimiento del aplicativo el uso de:

5.3.1. Versionamiento

El uso de un sistema que gestión el control de versiones del aplicativo implementado es un punto favorable al momento de dar soporte y mantenimiento del mismo. Los desarrolladores deben utilizar en lo posible versionamiento del código con el uso de servicios existentes como GitHub que aparte de ser un repositorio de código ayuda de sobremanera el versionamiento del mismo facilitando la gestión del aplicativo en su conjunto de funcionalidades definidas denominadas por un nombre y número de versión.

En la presente investigación se utilizó para los dos escenarios el repositorio con cuenta gratuita de GitHub donde actualmente reposan los proyectos en las siguientes urls:

MyApi Servicio Seguro - https://github.com/efaby/myApi , (***Figura 8-5***)

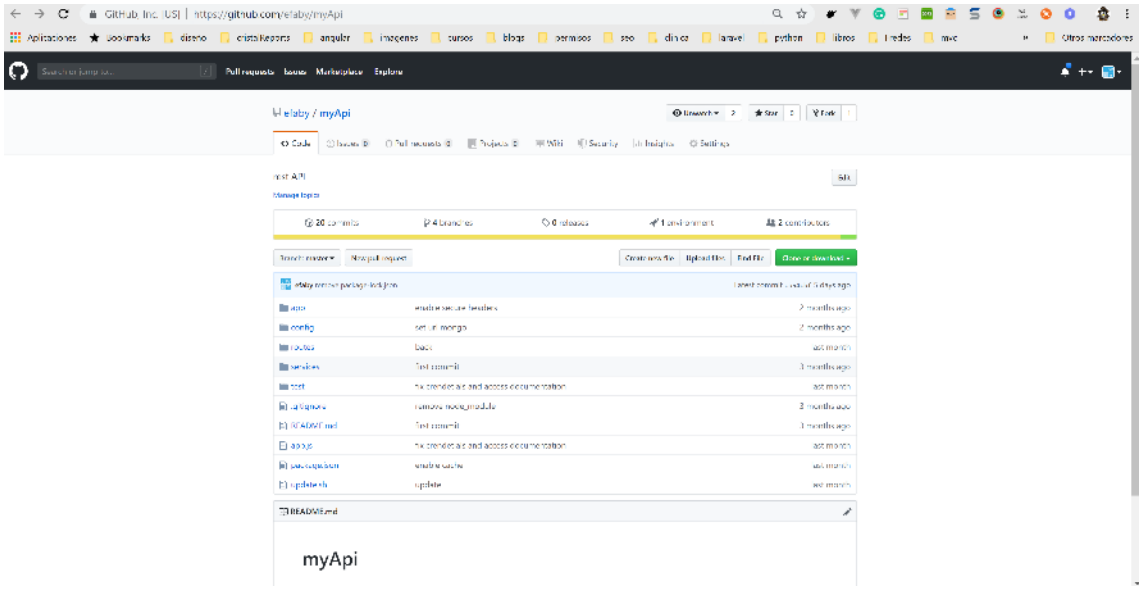


Figura 8-5: Repositorio de Código de Servicio Seguro
Realizado por: Villa Fabián, 2019

ApiInit Servicio Base - https://github.com/efaby/apiInit, (***Figura 9-5***)

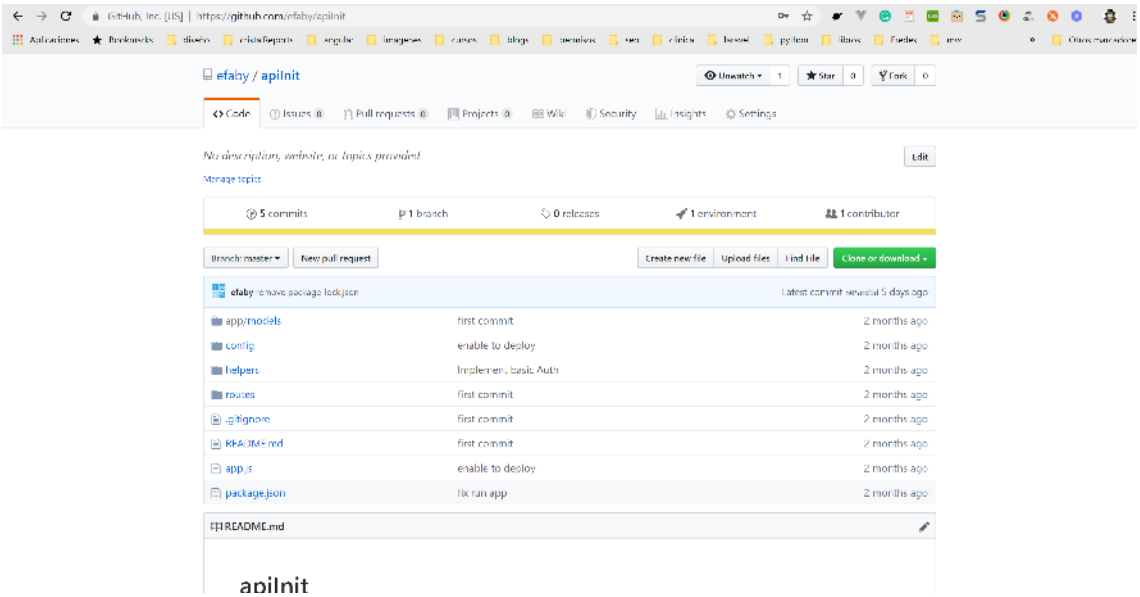


Figura 9-5: Repositorio de Código de Servicio Base
Realizado por: Villa Fabián, 2019

5.3.2. Uso de Frameworks

Antes de la implementación de un aplicativo es recomendable el investigar qué cosas ya están hechas o implementadas para optimizar recursos y obtener mejores resultados. En el mundo del desarrollo existen conjuntos de librerías implementadas en un determinado lenguaje de programación que cumplen funciones específicas denominados Frameworks.

Los Frameworks ayudan a la implementación de sistemas otorgando una serie de procesos ya implementados y listos a utilizarse. El desarrollador solo hace uso de estas funcionalidades y cumple con su objetivo, aunque mucha de las veces le toca implementar algunos procesos específicos, pero sobre una base sólida ya implementada.

En la presente investigación se implementó con la ayuda de framework Expressjs que ayuda a crear Servicios Web RESTFul de una manera rápida con el uso de librerías nodejs apropiadamente documentadas y seguras, simplificando la implementación del servicio.

5.3.3. Disponibilidad del Aplicativo

Para que un Servicio Web RESTFul sea lo suficientemente optimo y sea utilizado se requiere de una característica importante “debe siempre estar disponible”. Para ello es necesario proveer de una sólida infraestructura necesaria para que el servicio funcione en óptimas condiciones. Esto se puede lograr dependiendo del presupuesto de la organización que implementa el servicio.

Existen diferentes maneras de levantar un entorno de ejecución de un Servicio Web RESTFul entre ellas esta que se disponga de los recursos para adquirir la infraestructura física y posteriormente implementar la configuración necesaria. Otra manera que ayuda mucho es la de alquilar servicios de infraestructura en la nube.

Existen muchas empresas que alquilan dichas infraestructuras pasa los servicios facilitando de sobremanera la publicación de los aplicativos desarrollados. Al alquilar estos servicios de infraestructura se da paso a que sean ellos los que se encarguen del mantenimiento y de la operatividad del servidor donde se ejecuta el aplicativo y con una adecuada gestión de errores y sobretodo una adecuada implementación se garantiza que el servicio implementado va a estar siempre disponible.

En esta investigación se hizo uso de dos proveedores de servicios que trabajan de diferente manera para comprobar hasta que punto se puede gestionar la configuración del servidor y poder añadir seguridad a nivel de infraestructura.

Para el aplicativo **init-api** se utilizó el servicio de Heroku (***Figura 10-5***), que permite de manera muy fácil e intuitiva el despliegue del aplicativo por medio de su portal web y hace una conexión directa hacia GitHub permitiendo enlazar el aplicativo versionado al entorno de producción.

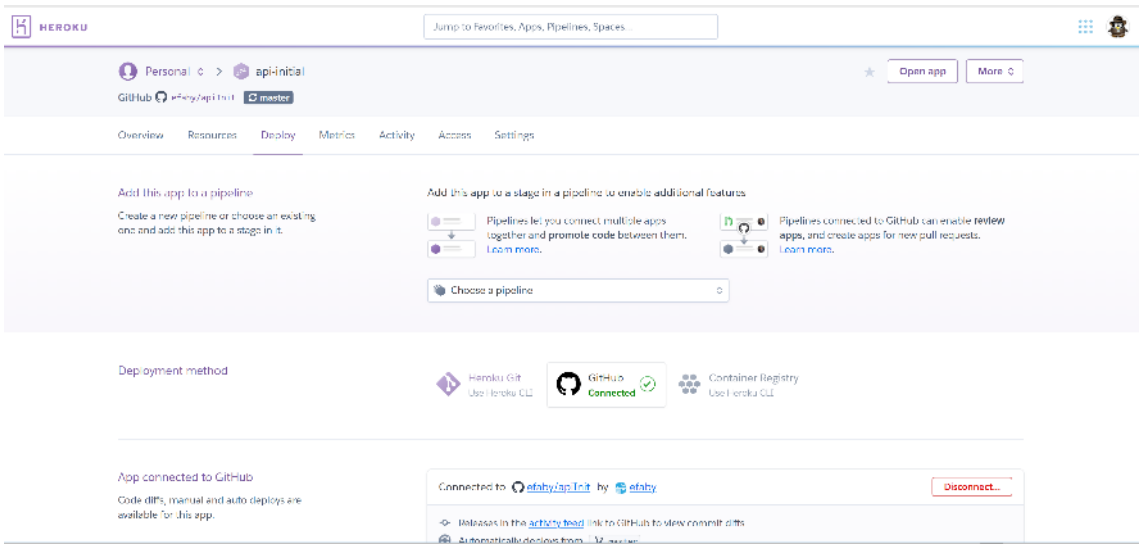


Figura 10-5: Dashboard del Heroku para el aplicativo init-api
Realizado por: Villa Fabián, 2019

Para el aplicativo my-api se utilizó el servicio de DigitalOcean que permite crear Servidores Virtuales Privados para configurarlos desde cero, y poder ahí sin restricciones configurar tanto el servidor como el aplicativo de una manera segura. Para esta opción requiere de mayor

conocimiento. En la siguiente figura podemos observar el despliegue de la aplicación desde GitHub por medio de la consola de comandos (***Figura 11-5***).

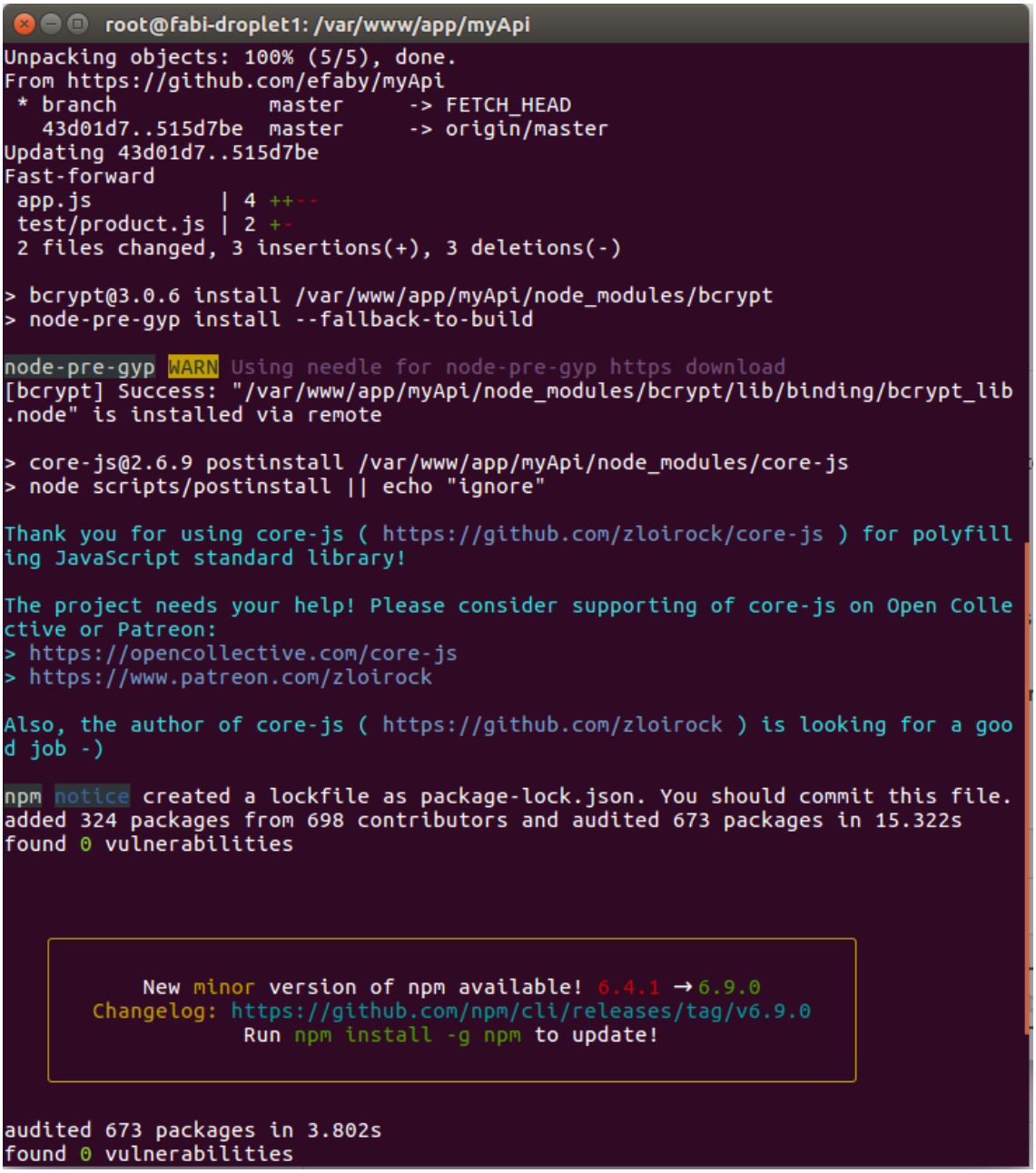


Figura 11-5: Despliegue del aplicativo seguro desde la consola de un Doplet en digital Ocean
Realizado por: Villa Fabián, 2019

CONCLUSIONES

- Se ha estudiado los diferentes métodos existentes de implementación de los Servicios Web RESTFul definidos en REST Security Cheat Sheet de OWASP, analizando cada uno de ellos para poder aplicarlos, tomados de forma no probabilística, partiendo desde la más transcendental en cuanto a seguridad usando la tecnología de nodejs para el desarrollo del nuevo servicio.

- Al analizar las vulnerabilidades más conocidas en los Servicios Web RESTFul implementados, se puede citar un mal uso de intercambio de credenciales en las peticiones, exposición sensible de data en las cabeceras, controles de acceso al servicio definido por los métodos existentes.

- Se implementó la propuesta de un método de mejores prácticas para la creación de los Servicios Web RESTFul con la cual se desarrolló el Escenario 2 y se obtuvo un 88.89% del nivel de satisfacción de la aplicación del método propuesto, sobre el Escenario 1 que se implementó con técnicas existentes.

- El uso de herramientas de pentesting como Vooki permitió analizar las vulnerabilidades expuestas sobre los dos escenarios, obteniendo así una reducción del 100% en vulnerabilidades Altas y Medias y un 50% en las bajas, obteniéndose un resultado de 90% de reducción total de vulnerabilidades en el escenario 2 implementado con el método propuesto.

RECOMENDACIONES

- El uso del método propuesto para el desarrollo de Servicios Web RESTFul, cuando se tenga la intención de crear un servicio desde cero, propone tomar en consideración aspectos como validaciones protección de urls, pruebas unitarias y versionamiento del código, además se recomienda hacer uso de patrones de arquitecturas de desarrollo de software como el MVC que permite tener el código organizado y de fácil mantenimiento.

- Definir el lenguaje de programación más óptimo para este tipo de servicios, ya que puede ayudar mitigando algunas consideraciones de seguridad con el uso de frameworks ya implementados, en nuestro caso de estudio se utilizó node js con su framework Express, sin embargo, existen otros frameworks con mejores características como es el caso de Loopback.

- La implementación de la documentación, ya que es un tema importante que a veces se deja de lado. Una documentación permite que el aplicativo pueda ser utilizado de una manera óptima y además se pueda aprovechar al máximo los servicios que este brinda.

- Implementar un Servicio Web RESTFul Seguro no solo depende de tener métodos y técnicas seguras de desarrollo, también implica tener una adecuada configuración del ambiente donde el aplicativo va a trabajar, de eso depende que el aplicativo funcione adecuadamente, y no siempre se cuenta con los recursos para tener una infraestructura propia sin embargo en esta investigación se citan dos servicios de infraestructura en la nube que puedan ayudar con cómodos planes de pago y que brindan un óptimo soporte técnico que ayuda a mantener el aplicativo funcional

- Ningún sistema o software va a ser 100% seguro, por más que se aplique técnicas o los mejores métodos, cada día se descubrirán nuevas vulnerabilidades que comprometan a los sistemas, es por ello que se debe periódicamente realizar mantenimientos, monitoreo y actualizaciones garantizando el óptimo funcionamiento del aplicativo.

- Como trabajo futuro se puede elaborar un método de mejores prácticas de configuración de la infraestructura que contemple configuración de dominios, certificados digitales, reglas de firewall y un middleware que permita filtrar accesos y número de peticiones junto con una configuración de servidores OAuth versión 2 que permita gestionar la autorización y autenticación de los aplicativos clientes.

BIBLIOGRAFÍA

Arcuri, A. (2019). RESTful API Automated Test Case Generation with EvoMaster. *ACM Trans. Softw. Eng. Methodol.*, 3:1--3:37.

Atlidakis, V., Godefroid, P., & Polishchuk, M. (2018). REST-ler: Automatic Intelligent REST API Fuzzing.

Bojinov, V. (2015). *RESTful Web API Design with Node.js.* Livery Place.

De Backere, F., Hanssens, B., Heynssens, R., Houthooft, R., Zuliani, A., Verstichel, S., . . . De Turck, F. (2014). Design of a security mechanism for RESTful Web Service communication through mobile clients. *IEEE Network Operations and Management Symposium (NOMS)*, 1-6.

DigitalOcean, LLC. (s.f.). *Digital Ocean.* Obtenido de https://www.digitalocean.com

Docker Inc. (s.f.). Obtenido de https://www.docker.com/

Doelling, C. (2018). FOUNDATIONS OF RESTful Architecture. *DZone, Inc.*, 1-8.

Doglio, F. (2015). *Pro REST API Development with Node.js.* La Paz, Canelones: Editorial Board.

Ed-douibi, H., Cánovas, J. L., Gómez, A., Tisi, M., & Cabot, J. (2016). Generation of RESTful APIs from Models. *ACM/SIGAPP Symposium on Applied Computing*.

GeoGebra. (2019). *Geogebra.* Obtenido de https://www.geogebra.org/m/YQCfcR2J

Giessler, P., Gebhart, M., Sarancin, D., Steinegger, R., & Abeck, S. (2015). Best Practices for the Design of RESTful Web Services. *Proceedings - International Conference on Software Engineering*, 392-397.

GitHub, Inc. (s.f.). *Git Hub*. Obtenido de https://github.com/

Hamad, H., Saad, M., & Abed, R. (2010). Performance Evaluation of RESTful Web Services for Mobile Devices. *International Arab Journal of e-Technology*, 72-78.

Lee, S., Jo, J.-Y., & Kim, Y. (2015). Method for secure RESTful web service. *IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, (págs. 77-81). Las vegas.

Masood, A., & Java, J. (2015). Static Analysis for Web Service Security – Tools & Techniques for a Secure Development Life Cycle. *IEEE International Symposium on Technologies for Homeland Security (HST)*, 1-6.

MongoDB, Inc. (2019). *Mongo DB*. Obtenido de https://www.mongodb.com/

Netquest. (12 de Diciembre de 2014). *Netquest*. Obtenido de https://www.netquest.com/blog/es/la-escala-de-likert-que-es-y-como-utilizarla

Neumann, A., Laranjeiro, N., & Bernardino, J. (2018). An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*.

Prasher, N. (2018). Security Assurance of REST API based.

Salesforce. (s.f.). *Heroku*. Obtenido de https://www.heroku.com/

Salibindla, J. (2018). Microservices API Security. *International Journal of Engineering Research & Technology (IJERT)*, 277-281.

Santos, T., & Serrao, C. (2016). Secure Javascript Object Notation (SecJSON). *The 11th International Conference for Internet Technology and Secured Transactions*, (págs. 329-334.). Barcelona.

Segura, S., Parejo, J., Troya, J., & Ruiz-Cortés, A. (2018). Metamorphic Testing of RESTful Web APIs. *IEEE Transactions on Software Engineering*, 1083-1099.

Sublime HQ Pty Ltd. (s.f.). *Sublime Text*. Obtenido de https://www.sublimetext.com/

Tarkowska, A., Carvalho-Silva, D., Cook, C. E., Turner, E., Finn, R. D., & Yates, A. D. (2018). Eleven quick tips to build a usable REST API for. *PLOS Computational Biology*.

The Linux Foundation. (s.f.). *Node JS*. Obtenido de https://nodejs.org/es/

Universidad Carlos III de Madrid - Departamento de Estadistica. (s.f.). Obtenido de http://www.est.uc3m.es/esp/nueva_docencia/getafe/ciencias_estadisticas/TecnicasInferenciaEstadistica/tablachicuadrado.pdf

Yarygina, T. (2017). RESTful Is Not Secure. En Springer, *Communications in Computer and Information Science,* (págs. 141-155). Singapore: Editorial Board.

ANEXOS

ANEXO A. Código principal del aplicativo api-init. Archivo app.js contiene la integración de todos los módulos del aplicativo como el uso de express, integración con mongodb, definición de rutas, y método de autenticación Basic.

```
const Express = require("express");
const Mongoose = require("mongoose");
const BodyParser = require("body-parser");
const mongoose = require('mongoose');
const routes = require("./routes/routes.js");
const config = require("./config/config");
const basicAuth = require('./helpers/basicAuth');
const errorHandler = require('./helpers/errorHandler');


let app = Express();
const router = Express.Router();

app.use(BodyParser.json());
app.use(BodyParser.urlencoded({ extended: true }));
mongoose.connect(config.database, { useNewUrlParser: true });

routes(router);
app.use(basicAuth);

app.use("/api", router);
app.get('*', function(req, res) { res.status(404).send({message: 'page not found!'}) });
app.use(errorHandler);

const server = app.listen(process.env.PORT || 5000, () => {
    console.log("Base Api RestFull is running on port", server.address().port);
});
```

ANEXO B. Código principal del aplicativo my-api aplicado los métodos de mejores prácticas. Archivo app.js contiene la integración de todos los módulos del aplicativo como el uso de express, integración con mongodb, definición de rutas, método de autenticación JWT, configuración de cabeceras seguras, definición de documentación y controles de acceso.

```
const express = require("express");
const bodyParser = require("body-parser");
const morgan = require("morgan");
const routes = require("./routes/routes.js");
const mongoose    = require("mongoose");
const cors = require("cors");
const config = require("./config/config");
const swaggerJSDoc = require('swagger-jsdoc');
const swaggerUi = require('swagger-ui-express');
const helmet = require('helmet');
const rateLimit = require("express-rate-limit");
const noCache = require('nocache')

const app = express();
const router = express.Router();

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs,
  message: "Too many accounts created from this IP, please try again after an hour"
});

app.use(helmet());
app.use(noCache());
const corsOptions = {
  origin: 'https://yourdomain.com'
}
app.use(cors());
app.use(morgan("dev"));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
mongoose.connect(config.database, { useNewUrlParser: true });
routes(router);
app.use("/api/", limiter);
app.use("/api", router);

// -- setup up swagger-jsdoc --
const swaggerDefinition = {
  openapi: "3.0.0",
    info: {
      title: 'My Secure API',
      version: 'v1',
      description: 'A Secure Api RestFull',
  },
    host: 'localhost:5001/api',
    basePath: '/',
    security: {
      bearerAuth: [],
    },
};

const options = {
  swaggerDefinition,
  apis: ['./routes/*.js'],
};
const swaggerSpec = swaggerJSDoc(options);

// -- routes for docs and generated swagger spec --

app.get("/ping", (req, res) => {
    res.status(200).send("I'm alive");
});

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));
app.get('*', function(req, res) { res.status(404).send({message: 'page not found!'}) });
const server = app.listen(process.env.PORT || 5001, () => {
    console.log("Api RestFull is running on port", server.address().port);
});

module.exports = app;
```